Parallel CFD 2009

21st International Conference on
Parallel Computational Fluid Dynamics

May 18–22, 2009, Moffett Field, California, USA

Preface

On behalf of the organizers, it is my distinct pleasure to welcome you to the 21st International Conference on Parallel Computational Fluid Dynamics (ParCFD 2009) here in Moffett Field, California—home of NASA's renowned Ames Research Center. First convened in Manhattan Beach, California in 1989, this is the 20th anniversary of the ParCFD conference, an annual meeting dedicated to the discussion and dissemination of recent developments and applications of parallel computing in the field of CFD and related disciplines. Held at different venues around the world, ParCFD provides a global forum for the exchange of novel ideas related to the parallel solution of CFD problems from a spectrum of arenas. The conference brings together CFD researchers and practitioners from academia, government, and industry, enabling interactions between both experts and non-specialists in the field.

This year, the ParCFD conference features 48 contributed papers chosen from 82 submissions from 20 countries. Each extended abstract was reviewed by three experts from the Scientific Program Committee for originality, relevance, completeness, and clarity. In addition, the program includes six invited sessions of 24 papers and a poster session of 15 presentations. Together, these talks cover recent advances and emerging challenges in a variety of areas such as mechanical and aerospace engineering applications, parallel algorithms and solvers, large-scale supercomputing and application scaling, multidisciplinary design optimization, unstructured overset grid methods, and parallel meshfree techniques. Six internationally recognized experts will deliver plenary lectures on high-performance computational engineering, high-fidelity simulations using high-order methods, real-time in-flight icing simulations, block-structured Cartesian grid techniques, parallel computational models for exascale architectures, and future directions in high-performance computing. A panel of experts will explore the question of how CFD applications can successfully exploit petaflops-scale computing systems and beyond. Finally, a tutorial on programming models for emerging multi-core architectures will educate attendees on how these systems, with increasing numbers of cores, can be effectively exploited.

This book consists of all invited and contributed abstracts and posters presented at ParCFD 2009. A DVD-ROM containing the full papers will be mailed later this year to all conference attendees. I would like to thank all authors and speakers for their contributions, and the members of the Scientific Program Committee for providing constant guidance and timely evaluation of all the submissions. I am also indebted to the invited speakers, invited session organizers, panel members, and tutorial instructors who helped make this a truly high-quality meeting. I am grateful to all the sponsors for their financial support without which the conference would not have been possible. Finally, my sincere thanks to the Local Organizing Committee whose long hours, can-do attitude, and constant cooperation made this conference a success.

Dr. Rupak Biswas
ParCFD 2009 Conference Chair

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

2009 ParCFD COMMITTEES

INTERNATIONAL SCIENTIFIC AND ORGANIZING COMMITTEE

Ramesh K. Agarwal	Washington University in St. Louis, USA
Rupak Biswas	NASA Ames Research Center, USA
Gunther Brenner	Technical University of Clausthal, Germany
Boris Chetverushkin	Russian Academy of Sciences, Russia
Anil Deane	University of Maryland, USA
Akin Ecer	IUPUI, USA
David R. Emerson	Daresbury Laboratory, UK
Pat Fox	IUPUI, USA
Marc Garbey	University of Houston, USA
Ulgen Gulcat	Istanbul Technical University, Turkey
David E. Keyes	Columbia University, USA
Trond Kvamsdal	SINTEF, Norway
Jang-Hyuk Kwon	Korea Advanced Institute of Science and Technology, Korea
Chao-An Lin	National Tsing Hua University, Taiwan
Kenichi Matsuno	Kyoto Institute of Technology, Japan
James McDonough	University of Kentucky, USA
Sergey Peigin	Israel Aircraft Industries, Israel
Jacques Periaux	CIMNE/UPC Barcelona, France
Nobuyuki Satofuka	University of Shiga Prefecture, Japan
Pasquale Schiano	Centro Italiano Ricerche Aerospziali CIRA, Italy
Suga A. Sugavanam	IBM, USA
Damien Tromeur-Dervout	University of Claude Bernard Lyon 1, France
Ismail H. Tuncer	Middle East Technical University, Turkey
Gerhard Wellein	University of Erlangen, Germany
Peter Wilders	Delft University of Technology, The Netherlands
Gabriel Winter	University of Las Palmas, Spain

2009 ParCFD COMMITTEES

TECHNICAL PROGRAM COMMITTEE

Hasan Akay	IUPUI, USA
Shahrouz Aliabadi	Jackson State University, USA
Ron Bailey	NASA Ames Research Center, USA
Bryan Biegel	NASA Ames Research Center, USA
Rupak Biswas	NASA Ames Research Center, USA
Akin Ecer	IUPUI, USA
Tom Edwards	NASA Ames Research Center, USA
Charbel Farhat	Stanford University, USA
Terry Holst	NASA Ames Research Center, USA
Dochan Kwak	NASA Ames Research Center, USA
Charles Nietubicz	Army Research Lab, USA
Bharat Soni	University of Alabama at Birmingham, USA
Eugene Tu	NASA Ames Research Center, USA
Erdal Yilmaz	Jackson State University, USA

LOCAL ORGANIZING COMMITTEE

Holly Amundson	NASA Ames Research Center, USA
Ron Bailey	NASA Ames Research Center, USA
Rupak Biswas	NASA Ames Research Center, USA
Akin Ecer	IUPUI, USA
John Hardman	NASA Ames Research Center, USA
Gina Morello	NASA Ames Research Center, USA
Doug Walker	NASA Ames Research Center, USA

CONFERENCE PANEL

Conference Panel: Petaflops and Beyond

Moderator: Ron Bailey, NASA Ames Research Center

Panelists:

- *Michael Aftosmis, NASA Ames Research Center*
- *David Emerson, Daresbury Laboratory, UK*
- *John Grosh, Lawrence Livermore National Laboratory*
- *John Shalf, Lawrence Berkeley National Laboratory*
- *Suga A. Sugavanam, IBM Systems & Technology Group*

The purpose of this panel is to explore the question of how CFD applications can exploit computing systems with performance potential of petaflops and beyond. Due to limits on switching speeds and power consumption, high-performance computers are moving into a new era dominated by many-core chips and heterogeneous architectures. As their speeds advance beyond petaflops and toward exaflops, computers can be expected to reach one million cores and perhaps more. Thus, it seems clear that new programming approaches will be needed, and perhaps, new algorithms if we are to realize significant real performance increases for CFD applications.

Suggested Questions to the Panel:

- What will future architectures and systems look like as performance extends from petaflops toward exaflops?
- What are the dominant CFD applications that could benefit from petaflops-and-beyond computing?
- What are the algorithmic and programming challenges that must be overcome to efficiently exploit petaflops-and-beyond hardware to solve CFD and other similar problems?
- What should the CFD community do to prepare to address challenges presented by petaflops-and-beyond systems, and to exploit their use?

INVITED SPEAKER BIOGRAPHIES

DIMITRIS DRIKAKIS, CRANFIELD UNIVERSITY

Dimitris Drikakis is professor of fluid mechanics and computational science, and head of the Aerospace Sciences Department at Cranfield University. In 2008, he was recognized by the William Penney Fellowship for his contribution to fluid mechanics and CFD, and their applications to aerospace and defense. Dimitris is Fellow of the Royal Aeronautical Society and the Institute of Nanotechnology. His research interests include computational fluid dynamics, fluid mechanics, heat transfer, aerodynamics, computational nanotechnology and materials modeling. Dimitris has co-authored, jointly with William Rider (Sandia National Labs), the book "High-Resolution Methods for Incompressible and Low Speed Flows," Springer, 2005. He is an associate editor of the Journal of Fluids Engineering, The Aeronautical Journal, and the Journal of Computational and Theoretical Nanoscience.

WAGDI HABASHI, MCGILL UNIVERSITY

Wagdi Habashi is a professor of mechanical engineering at McGill University in Montreal. He taught at the Stevens Institute of Technology in New Jersey for 2 years and for 25 years at Concordia University in Montreal prior to coming to McGill, where he is the NSERC-Bombardier-Bell Helicopter-CAE Industrial Research Chair (IRC) for Multi-disciplinary CFD, currently the only IRC among the Faculty. At McGill, Wagdi has established and is directing its Computational Fluid Dynamics Laboratory. He has also established the CLUMEQ (Consortium Laval-Université du Québec-McGill and Eastern Quebec) Supercomputer Center and was Director from 2000-2007, thrice funded by the Canada Foundation for Innovation and the Quebec Government ($60M for High Performance Computing).

DIMITRI MAVRIPLIS, UNIVERSITY OF WYOMING

Dimitri Mavriplis is Professor of Mechanical Engineering at the University of Wyoming. Prior to coming to Wyoming in 2003, Dr. Mavriplis spent 16 years at ICASE, NASA Langley Research Center where he developed unstructured mesh discretization and solution techniques for computational aerodynamics, many of which have been widely adopted by NASA and industry. During his career, he has worked on unstructured mesh generation methods and adaptive techniques, agglomeration multigrid solvers, time-implicit methods, and coupled aeroelastic problems. More recently he has been involved in the development of adjoint sensitivity analysis techniques for design-optimization and error estimation problems, as well as the development of high-order discontinuous Galerkin discretizations for aerospace engineering applications. He is a committee member and contributor to the AIAA drag prediction workshop series, and an AIAA Associate Fellow. Most recently, he has been involved in the advocacy of high-performance computing research and development for aerospace engineering disciplines.

INVITED SPEAKER BIOGRAPHIES

KAZUHIRO NAKAHASHI, TOHOKU UNIVERSITY

Kazuhiro Nakahashi is a professor within the department of aerospace engineering at Tohoku University in Sendai, Japan. Kazuhiro has developed CFD algorithms including a solution-adaptive grid method using tension-torsion spring analogy, FDM-FEM hybrid method, prismatic grid method, and overset unstructured grid method. During the last decade, he has been working on developing an unstructured grid CFD code named TAS (Tohoku University Aerodynamic Simulation). Currently, the TAS-code is used for aerospace applications at the Japan Aerospace Exploration Agency, industries, and universities throughout Japan. Kazuhiro is a fellow of the Japan Society of Mechanical Engineers and the Japan Society of Fluid Mechanics. He was elected president of the Japan Society for Aeronautical and Space Sciences for FY 2009.

HORST SIMON, LAWRENCE BERKELEY NATIONAL LABORATORY

Horst Simon is Associate Laboratory Director at Lawrence Berkeley National Laboratory for Computing Sciences, Division Director for the Computational Research Division, and Adjunct Professor of Computer Science at the University of California, Berkeley. His research interests are in the development of sparse matrix algorithms, algorithms for large-scale eigenvalue problems, and domain decomposition algorithms. His recursive spectral bisection algorithm is a breakthrough in parallel algorithms. He was also honored with the 1988 Gordon Bell Prize. He has served as a senior manager for Silicon Graphics, Computer Sciences Corporation, Boeing Computer Services, and has been a member of the faculty at the State University of New York. He is currently a member of the advisory boards of more than five research organizations located throughout the world, a member of many journal editorial boards, and one of four editors of the twice-yearly "TOP500" list of the world's most powerful computing systems.

THOMAS STERLING, LOUISIANA STATE UNIVERSITY

Dr. Thomas Sterling is a Professor of Computer Science at Louisiana State University, a Faculty Associate at California Institute of Technology, and a Distinguished Visiting Scientist at Oak Ridge National Laboratory. He received his Ph.D. as a Hertz Fellow from MIT in 1984. Dr. Sterling is probably best known as the "father" of Beowulf clusters and for his research on Petaflops computing architecture. He was one of several researchers to receive the Gordon Bell Prize for this work on Beowulf in 1997. In 1996, he started the inter-disciplinary HTMT project to conduct a detailed point design study of an innovative Petaflops architecture. He currently leads the MIND memory accelerator architecture project for scalable data-intensive computing and is an investigator on the DOE sponsored FastOS Project to develop a new generation of configurable light-weight parallel runtime software systems. Thomas is co-author of five books and holds six patents.

ABSTRACTS

Future Directions in High Performance Computing (HPC) 2009–2018

Horst D. Simon

Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
(Tel: 510-486-7377; e-mail: hdsimon@lbl.gov)

Abstract: Since about 2004 there has been a fundamental transition taking place in computing. The microprocessors clock speed improvement have leveled off, and future performance increases from processors will be realized through multi-core and many-core chips. This change in the basic building blocks for HPC has opened up the architecture discussion for future HPC platforms, and in 2008 we see vigorous experimentation with accelerators, GPUs, FPGAs, and embedded technology. HPC has not seen such a variety of new technology being explored since the early 1990s. In my talk I will explore what the multi-core revolution will mean for the future of HPC. I will use the very successful model of the HPC ecosystem and its important elements economic driver, system architecture, and programming model to explain how I think HPC will develop in the next five years. Several of the projects that are currently going on in Berkeley will be discussed in detail, since they will provide tools for the future productive use of supercomputers. They include auto-tuning, PGAS languages, and most importantly the Green Flash project, that attempts to find a new solution for energy efficient computing in the future.

A FRONTIER OF PARALLEL CFD: REAL-TIME IN-FLIGHT ICING SIMULATION OVER COMPLETE AIRCRAFT

Wagdi G. Habashi
Director, CFD Laboratory, Department of Mechanical Engineering
NSERC-J. Armand Bombardier Industrial Research Chair of Multidisciplinary CFD
McGill University, Montreal, QC, Canada H3A 2S6

ABSTRACT

With the power of supercomputers increasing exponentially, there is an insatiable need for more advanced multi-disciplinary aerospace CFD simulations. A particular current interest is the 3D viscous turbulent simulation of the highly nonlinear aspects of aero-icing. The applications of CFD in that field are literally light-years behind aerodynamics, with a significant number of users still mired in correlations, or 2D, inviscid, incompressible, and, yes, Panel Methods simulations! Thus, the disparity of tools between aerodynamics and icing departments within an organization leads to a disconnect that makes ice protection a downstream isolated process that is not an integral part of the aerodynamic behavior of an aerospace system (aircraft, rotorcraft, jet engine, UAV, etc.). While 3D RANS has been recently introduced, it is still considered computationally too demanding for industry when wide parametric studies for certification are required. In addition, not unlike the situation in aerodynamics say 20 years ago, the naysayers are at every corner claiming that CFD is not reliable and is of limited use.

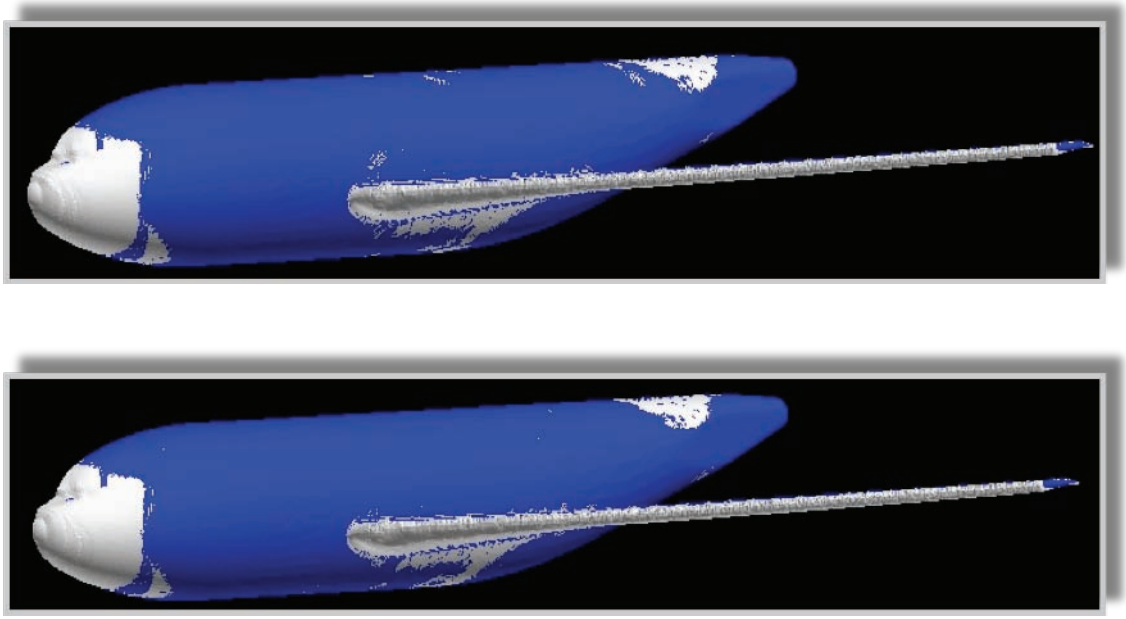
Another important chasm of in-flight icing is the lack of means for proper training of aircraft pilots as to the effects of ice on the aircraft behavior. That such training is too rudimentary has been brought up at many conferences by pilot representatives. In essence, in a simulator the effect of ice as transmitted to a pilot is to make the aircraft heavier and shift its center of gravity. In reality, pilots know better as different types of ice at different temperatures, rates of accretion, liquid water content and droplets diameter, coupled to the flight characteristics of the specific aircraft, can have dramatically different effects. Some aircraft are known to be so ice-sturdy that almost no amount of ice can affect them, while others can, at the first sign of roughness, exhibit dangerously sluggish behavior. Thus, it is primordial to analyze the icing and aircraft envelopes together and not separately, and bring CFD-based for the prediction of ice accretion rates and shapes and of performance degradation for the specific aircraft. The obvious difficulty, naturally, would be to carry out ice calculations in real time and not simply through a lookout table.

In order to make such compute-intensive simulations more affordable and eventually carried out in real-time, four steps would be needed:

1. The formulation of the icing problem as a truly unsteady phenomenon, and the derivation of new Navier-Stokes-like equations for each of its 5 successive facets: CFD of clean aircraft, droplet impingement, ice accretion, anti- and de-icing, and CFD of iced aircraft; repeat,
2. The speedup of all of these modules, through massive parallelization,
3. The formulation of Reduced Order Models to extract information from a limited number of high-fidelity runs to form a database for certification verification or for flight simulator data,
4. The parallelization of Reduced Order Models in order to yield results in real-time.

We will present reduced order models based on the "Proper Orthogonal Decomposition" method, and predict a wider swath of flow fields and ice shapes based on a limited number of "snapshots" obtained from complete high-fidelity (3D viscous, compressible, turbulent) CFD computations. Modes are extracted from these snapshots and used to reconstruct the CFD field, and/or the aerodynamic coefficients, and/or the ice shapes, for other conditions within the range. Each mode will be shown to represent an important and particular aspect of the physics of the problem. This reduces calculation times by two to three orders of magnitude from the full 3D ones, enabling a more complete map of the performance of an iced aircraft over a wide range of flight and weather conditions to be used in its certification and in pilot training. Work in progress will also be covered.

Ice on complete aircraft (CFD + Impingement + Accretion)
Above, high-fidelity CFD: days Below, POD: Seconds



Enabling Exascale through the ParalleX Paradigm

Thomas Sterling, Chirag Dekate

Center for Computation Technology &
Department of Computer Science,
Louisiana State University, Baton Rouge, LA 70808, USA
(Tel: 225-578-8982; e-mail: {tron, cdekate} @cct.lsu.edu)

Abstract: HPC is in a phase change with technology advances driving the development of new system and core architectures, programming models and languages, runtime and operating systems. A new model of computation is required to enable Exascale computing to address the challenges of scalability, latency, synchronization and scheduling overhead, and through put contention. This presentation will describe a paradigm being used for co-design of the layers of the system stack for systems of the end of the decade based on the innovative ParalleX model of computation. ParalleX is a message-driven multithreaded global address space strategy with advanced synchronization semantic constructs. Early results derived from experiments with a new runtime system supporting the ParalleX approach will be discussed.

Keywords: Exascale, ParalleX, High Performance Computing, Model of Computation.

1. INTRODUCTION

Throughout the 7 decades of the active field of the electronic digital stored program computer, technology for logic, memory, and communications has evolved providing changing opportunities for the structure, management, and application of high performance computing. For each transition in epoch, system design has experienced an event of punctuated equilibrium where a new class of computer architecture and programming model(s) would be created to exploit the new opportunities of the improved technology while compensating for their limitations through structural and usage innovation. Such dramatic changes constituted an HPC phase change. There have been 5 such HPC phase changes. Underlying each HPC phase change is by intent or implicitly a corresponding change in the foundation model of computation. A model of computation or alternatively an "execution model" is a paradigm that governs the structure, interrelationship, operation, management, and methodology or usage of the system. Other models also overlapped these such as dataflow, multithreaded, and systolic arrays that either found market niches or did not extend beyond the experimental proof of concept stage.

In the 6th phase (which may be designated the "heterogeneous multicore era") the technology drivers are as follows

1) The continued advances of device density (Moore's Law) with expected feature size of 8nm by 2018
2) Processor complexity for increased per core performance is exhausted past the point of diminishing returns.
3) The need for system performance pushing the bounds of capability by 3 orders of magnitude.
4) The suitability for an entirely new class of computation based on dynamic graphs necessary for sparse computations (e.g., adaptive mesh refinements) and informatics.

HPC Phases	Model of Computation
Accumulator/buffers	Sequential Issue
Registers/caches	Pipelined Execution
Pipelined ALU/register & memory-banks	Vector Processing
Array processors	SIMD Array
MPP/clusters	Communicating Sequential Processes

The sixth phase resulting from the new HPC phase change may be cast either as the Exascale era or the heterogeneous multicore era. The strategy of employing a new model of parallel computation as a conceptual framework for co-design of advanced systems to catapult HPC through this phase change is viable only if it serves as an enabler and tool to this end

3. GOALS OF A NEW MODEL OF PARALLEL COMPUTATION

Conventional practices employ message passing for scalable distributed memory systems. This limits the means by which distributed computation may be organized, controlled, and performed. An alternative is the use of message-

driven computation. This allows work to move to the data rather than always requiring that data be gathered to the work. Sometimes the later is appropriate when there is relatively little remote data needed while the context of execution is substantial and local. However, the opposite can also be true. The use of message-driven computation reduces the average amount of data movement, which also reduces power. Combined with the work-queue model, it provides innate system-wide latency hiding which is critical to overall system efficiency. One class of application that will particularly benefit from this is dynamic graph applications. In particular, an advanced class of message-driven computation may allow continuations to migrate across the system and the computation. This transfers the locus of control to the area of the system where the work needs to be conducted rather than trying to control it from a remote location. Conventional practices exploit avoidance, caching in the presence of temporal and spatial locality to memory, and prefetching. While these methods have some practical value, even in the future, they are restrictive and demand advanced structures and methods that overcome latency.

Finding parallelism in the computation is essential to exposing the billion-way parallelism required for Exascale computing. One area that may reveal significant more parallelism is through concurrent actions on the meta-data that define the structures of sparse data and directed graphs. Allowing the computing to traverse different paths through a graph at the same time exposes the inherent parallelism of the graph. But this requires a new way of synchronizing actions manipulating the graph. Informatics applications will be significantly determined by the meta-data of their global data structures. A form a data-directed execution will emerge where data structure and flow control merge.

Large distributed systems have orders of magnitude time variation between the most local actions and system-wide actions. Furthermore, due to a number of factors it is unlikely that other than the most constrained systems and programming methods can determine the time and ordering of all actions in the system. Local domains can bound these. These can be operated synchronously while the system overall may likely have to operate asynchronously. A future model of computation should support semantics and mechanisms that facilitate effective synchronous and asynchronous operation. This is necessary for retaining invariants of operation while existing in an asynchronous environment. It should provide runtime adaptive control for contention resolution, latency hiding, and load balancing. As an example of such mechanisms is the *futures* construct, first proposed in the late 70's, that allows computations to proceed at their best rate, supports anonymous producer-consumer computation, and managing eager versus lazy evaluation.

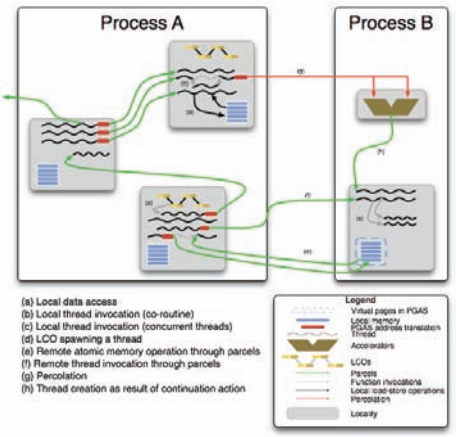
PGAS or partitioned global address space systems such as the early CRI T3E provides a good compromise providing a global address space but not guaranteeing cache coherence. However, PGAS fixes the location of a global variable to a single node. If load balancing or some other need for movement occurs, a virtual variable has to change its virtual name. Active Global Address Space may be an approach that circumvents the problems with cache coherence, exploits the possibility of PGAS, but goes further by permitting the movement of virtual variables in physical space without having to change its name. Conventionally, flow control is fixed to individual physical resources and their respective program counters. An opportunity exists to break this rigid approach and employ the abstraction of the "continuation" which is a construct that represents next actions to happen. A program counter is an instance of a continuation in the narrow sense. It indicates where the next operation of a process to be performed can be found. But when global parallelism is being harnessed for performance gain, then a global parallelism flow control construct is required. Such a continuation decouples the flow control from a fixed physical resource. It can be merged with the meta-data parallelism to provide the data directed execution. And it permits the migration of flow control to the position of the data that is to be worked on.

The principal goal of a new model of parallel computation is to serve as a conceptual framework to govern the co-design and interoperability of all the layers in the system stack from programming models down to core architectures and all of the hardware and software levels in between. Such a model will enable and support many distinct design points consistent within the discipline. Furthermore, the possibility of multiple models of computation adds an additional dimension to the richness of opportunity. A major goal for the new model is the dramatic increase in useful program and execution parallelism. The new model of computation has to expose, exploit, execute approximately billion-way parallelism to reduce execution time to solution, increase performance to Exascale capability, to hide latencies and dynamically employ available resources for high efficiency, and exploit the hundreds of millions of cores and two orders of magnitude or more threads and ILP. A second major goal is to enable the mitigation of local and global latency, which can impose delays of from hundreds to tens of thousands of cycles within the system and orders of magnitude before if secondary storage is included.

4. PARALLEX MODEL OF COMPUTATION

The ParalleX model of parallel computation is an example of one possible execution model that incorporates many of the opportunities. It supports message-driven work-queue execution in the context of an Active Global Address space. It provides for lightweight synchronization, continuation migration, and exposes meta-data parallelism.

Together these characteristics support dynamic optimization for best parallelism, load balancing, low overhead, and adaptive contention resolution. It is being enhanced to include a multi-phase micro-checkpointing model to deal with hardware failures and maintain continued operation. (It does not guarantee soft error or transient hardware error protection) ParalleX is also being enhanced for self-aware power management. Locality domains guarantee compound atomic operations, deliver bounded response time to issued instructions, and can ensure shared memory semantics. Synchronous operation within a locality provides low overhead for data access, thread creation, context switching, and memory access. Between localities actions are asynchronous using message-driven computation. Parcels are a form of active messages that include information about the destination object upon which it is to operate, the action that is to be performed, a set of operand values which may be as large as a page, and a continuation that indicates what should happen after the action has been completed. The Active Global Address Space provides a global shared memory name space. It is not cache coherent. Copy semantics are provided to allow distributed variables to maintain consistency on reads and writes. AGAS allows virtual variables to retain their virtual address even if they are moved between localities. Split-phase transactions are supported such that work performed on data on the locality may be passed to another locality when remote data needs to be processed.



This diagram illustrates many of the key semantic constructs and mechanisms of the ParalleX (PX) model of computation. The shaded squares represent physical localities. All operations within a locality are synchronous. All operations between localities are asynchronous. A process is a context that defines data and tasks to be performed on the localities to which it is assigned. A process may be allocated multiple localities; these not in any way mutually local to each other. More than one process can share a locality (not shown). Process A is allocated three localities while Process B is on one locality but also uses a separate physical execution unit. Processes can incorporate child processes (not shown). Large rectangular transparent boxes illustrate processes.

Basic tasks in the ParalleX model are conducted by "threads" which are partially ordered instructions working on a potentially infinite set of registers. This convenient abstraction minimizes the number of anti-dependencies to permit easy back end compilation to diverse ISA and data paths for different cores. In the diagram, black "squiggly" lines represent the threads. While a process may span multiple localities, a thread by definition resides on only one locality at a time and usually (but not necessarily) on a single locality throughout its lifetime. Threads are ephemeral: each is created at some point in time, performs its work, and terminates. A thread may be suspended but the ideal thread is relatively lightweight, works only on local data and terminates quickly. The effect of a thread may be manifest in several ways: 1) it changes local mutable state, 2) it creates sibling threads (or processes) that outlive it, or 3) it creates parcels that invoke remote threads (or processes) or change remote mutable state. Threads are first-class objects with names in the same name space as variables and objects. They can be manipulated by threads potentially of other processes. Threads can be suspended to form objects called "depleted threads". A depleted thread contains all the private state of an active thread but takes up no physical execution resources other than a little main memory space. A depleted thread is a first class object and has the same virtual address of the original thread from which it is derived. Therefore, it can be accessed; its state modified; and, it can cause a new thread with the original name to be created (equivalent to reactivating the old thread). Threads perform almost all the work of the program. The exception is the parcel that directly effects remote state without invoking a thread.

Parcels are a class of active messages. They are illustrated by the green lines that connect between localities. Parcels support semantics at remote localities that threads can perform directly within their resident locality. These are creating remote threads or processes, changing remote state through compound atomic operations, or manipulating threads or processes directly. A parcel carries four types of information: a) the name (virtual address) of the object upon which it is to operate, b) the action to be performed, c) a set of operand values, and d) a continuation specifying what is to happen upon completion of the specified action. The actions can be thread invocations, process instantiations, or primitive compound atomic operations on virtual data or physical devices with physical addresses. Parcels support asynchronous operation of the ParalleX environment. Parcels can arrive at any time and order of arrival is not guaranteed. They can be relatively short with little or no operand values, or be very long when used to move blocks of data. Parcels are not first class objects and do not have virtual names. Locally at the source and destination localities parcels do display handles for local manipulation. Parcels belong to a global object called the "parcel set". A special form of the parcel is used for a methodology called percolation. Percolation moves the entire

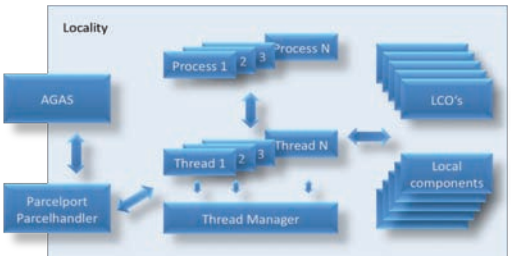
work footprint of a thread to a remote physical execution site. It is illustrated by the red lines between thread and functional unit. By foot print it is meant to include the instructions of the thread, the stack frame of the thread, the input arguments used for the thread, and any other information required to fully instantiate the thread on the remote execution site. Percolation is used to make efficient use of relatively precious resources like accelerators, GPUs, FPGAs, or expensive streaming processors. It does so by assuming the overhead and hiding the latencies that such a unit would otherwise have to incur. Thus percolation supports prestaging of execution information. Percolation is also used to dynamically allocate new resources within a system to an application program.

A critical element to the management and coordination of global parallel flow control is the Local Control Object of "LCO" shown as small yellow/brown-green splotches. An LCO is a synchronization object in the object-oriented sense. It has associated with it both state and methods that operate upon the state. LCOs are first-class objects in that they have names that are virtual addresses like other variables. The purpose of the LCO is to coordinate global flow control by means of a number of different types of synchronization mechanisms from simple mutex elements to sophisticated dataflow templates and *futures* constructs. An LCO is accessed like any other object locally by a thread or remotely by a parcel possibly changing its state. An LCO includes one or more criteria which if satisfied will result in the instantiation of a thread. LCOs can also buffer incoming requests until such a condition can be satisfied. This may include the availability of physical resources. The LCO is the embodiment of a continuation. The continuation field of a parcel with some exceptions is a reference to one or more LCOs.

The semantics of thread to thread and thread to data interrelationships of the ParalleX model of computation are symmetric with respect to intra-locality synchronous operation and inter-locality asynchronous operation using parcels or possibly percolation.

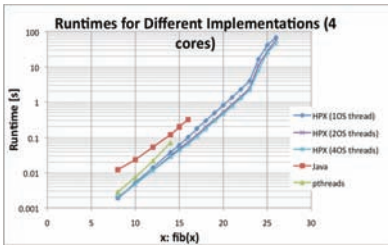
4. PARALLEX: IMPLEMENTATION & INITIAL RESULTS

Implementation of the ParalleX model of computation involves many system levels. Among these are the runtime system which is created for a given application when the application is started and is terminated when the application finishes. All state and activities are performed within the contexts of parallel processes. A process may be assigned one or more localities upon which it stores its state variables and that perform its threads. Processes may share localities. The process that employs all localities assigned to the application program and for which all other processes and threads are descendents is called process "main".



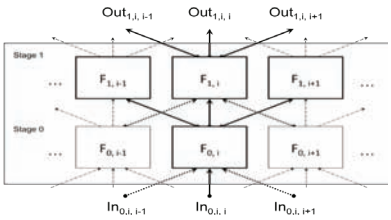
All named objects live in global memory and are accessible through the Active Global Address Space manager. The runtime system maintains the address map on behalf of the application and with the support of the operating system. First class objects including program variables are allocated space in memory organized according to the hierarchy of processes. The runtime system maintains the AGAS map. In the case of linked data structures like dynamic graphs, it also maintains virtual to physical address translations cached in the data structure itself. These entry are updated upon data migration. Within a locality are many threads in different states as discussed earlier. The runtime system includes the thread manager, which is responsible for all aspects of thread operation from their beginning in the start state to their termination in the end state. The thread manager controls the state transition of every thread, schedules the threads for execution on the physical resources, preempts threads on the physical resources when other threads require access to such resources, supervises the thread queue hierarchy, and creates new threads from parcels, local control objects, and from parent threads.

The runtime system also controls local control objects (LCOs). Like transactional memory semantics LCOs require guaranteed compound atomic operations to ensure resolution of concurrent incident accesses. The LCOs determine the conditions for creation of new threads and the runtime system updates LCOs and when ready creates the new threads. The runtime system is also responsible for the management of parcels. The runtime system supports the parcel handler that keeps waiting parcels in queue until there is room on the locality to serve them as new threads. It also out queues new parcels created by threads on the locality until there is sufficient available network bandwidth to send them out. The runtime system is distributed among all localities associated with a given application. It contributes to distributed functions like AGAS maintenance and parallel process support. Local software components are the building blocks, implemented in C++, from which these distributed runtime system functions are comprised.

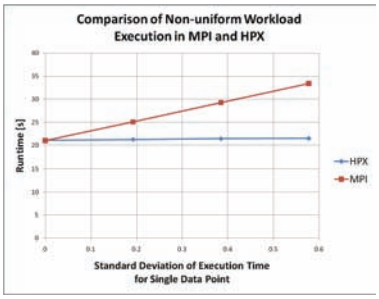


As a simple but challenging example of the operation of the HPX runtime system, a problem running millions of cores was performed for the recursive Fibonacci sequence. This is a very poor way to compute Fibonacci numbers but a great way to generate many very lightweight threads. In effect, this is one of the more challenging toy applications creating an enormous amount of rather useless parallelism in terms of number of concurrent threads. This first diagram shows a single physical execution unit at a time running three different versions of fib: using Java, Pthreads, and HPX. The graph shows time to completion with respect to the Fibonacci number. Interestingly, the Java version exhibited the worst performance. Pthreads under Linux was significantly better than Java. But the HPX threads was about twice as good as the Pthreads version. More importantly, the HPX runtime system was able to handle many more concurrent threads than Pthreads, which was the worst or the Java system. This demonstrates even at an early stage of developments that HPX runtime thread management is superior by a significant degree that other conventional practices of the Java virtual machine or the Unix Pthreads support. Using two cores to the previous experiment shows good scaling for the HPX runtime system with response time becoming half of the single core runtime while maintaining the same number of concurrent threads, still far greater than the Java virtual engine or the Pthreads OS implementation.

A more important test case using ParalleX and the HPX runtime system is one in which one of the key properties of the model is exposed. ParalleX can use dataflow like synchronization and control flow eliminating the over constraining use of global barriers. Although somewhat simplified, this problem comes from a numerical relativity adaptive mesh refinement problem. Many different threads from the inner loops of the problem are concurrent but the length of execution can vary by well over an order of magnitude. Further, for any one thread of the next iteration only requires that three of the threads from the previous iteration be completed. ParalleX offers the opportunity to start future threads when only the precedence threads are completed and also allows for dynamic thread scheduling. These two related features are captured by this experiment differentiating it from the conventional use of global barriers. In this traditional approach, all threads of one iteration need to be completed to satisfy the global barrier before any of the threads of the next iteration can begin; an over-constrained condition.



The experiment uses a synthetic set of inner loop execution times retaining the same mean but increasing the variance of the set of threads. The diagram above compares the HPX implementation shown in blue with the comparable implementation in MPI with global barriers shown in red. The length of time between successive global barriers for the MPI implementation is determined by the slowest thread even if many threads complete in a small fraction of this time. The HPX implementation only requires a few threads to complete to launch a thread in the next iteration. As the variance increases, the length of execution of the slowest thread for the MPI implementation grows, increasing the overall application execution time. In comparison, the HPX runtime system dynamically selects the next available thread to be performed and overlaps these with threads of the next iteration whose precedence constraints have been satisfied. As shown in the diagram, the runtime of the HPX execution remains relatively constant even as the variance increases, the mean remaining constant, while the runtime for the MPI version grows proportionally. This demonstrates an important performance and efficiency opportunity of the ParalleX model of computation compared to the MPI model using global barriers.



REFERENCES

1. Kaiser, H., Brodowicz, M., Sterling, T. (2009). Advanced Parallel Execution Model for Scaling Impaired Applications, *Submitted to International Conference on Parallel processing ICPP*.
2. Brandt, S., Stark, D., Dekate, C., Sterling, T., (2009) PXI: A ParalleX programming interfaces. *Submitted to International Workshop on Parallel Programming Models and Systems Software for High-End Computing P2S2*.

High-fidelity CFD simulations of shock physics, instabilities, transition and turbulence using high-order methods and parallel computing

D. Drikakis*, M. Hahn, B. Thornber and E. Shapiro

Fluid Mechanics and Computational Science (FMaCS) Group, Department of Aerospace Sciences, Cranfield University, Cranfield, Bedfordshire, MK43 0AL, United Kingdom.

**Invited speaker: D. Drikakis, e-mail: d.drikakis@cranfield.ac.uk*

Abstract: The paper presents advances in modelling and simulation of complex transitional and turbulent flows in the framework of large eddy simulation in conjunction with parallel computing and high-resolution and high-order methods. In particular, the paper reports results from computational studies of shock-waves induced instabilities and turbulence, mixing layers and flows around swept wings, including comparisons with experimental data. Parallel efficiency studies from the implementation of compressible CFD methods in the computational platform HIRECOM show good scalability up to 1024 processors, thus enabling turbulent flow simulations of up to $3 \cdot 10^9$ points.

Keywords: Implicit Large Eddy Simulation, high-resolution methods, transition, turbulence, shock physics

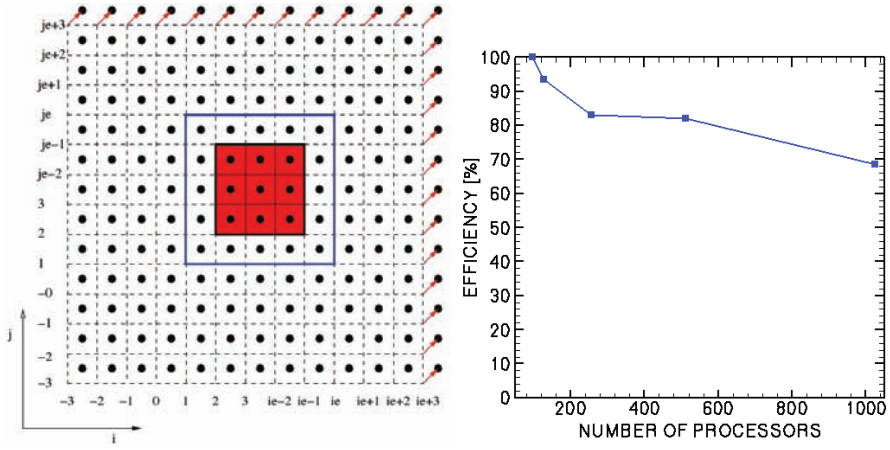
1. INTRODUCTION

Advances in hardware development have led to increased utilisation of high performance parallel platforms in scientific computing. The Top 500 list published in November 2008 contains two entries delivering over 1 Petaflop/s of sustained performance. Despite the advances in hardware and software development, computational science and engineering problems continue to push the boundaries of computational power available. Simulation of transitional and turbulent flows is a longstanding example, which is relevant to a range of industries with applications encountered in aerospace and defence, chemical, and energy sectors, to name just a few.

The turbulent flow theory proposed by Kolmogorov in the 1940s (see, for example, [1]) allows us to clearly identify the computational requirements of an idealised time-accurate fully resolved Direct Numerical Simulation (DNS) of a turbulent flow. The total number of points in time/space required is proportional to Re^3 per unit hypercube, where Re is the Reynolds number of the flow. Taking a typical cruise Reynolds number for a passenger aircraft equal to 10^8, we can estimate that even if a single point in space-time continuum requires a single floating point operation, the most powerful supercomputer in the world can simulate a unit space/time hypercube within approximately 200 years. These stringent computational requirements make DNS impractical in high Reynolds number applications. Indeed the largest DNS simulation performed to date is the isotropic turbulence simulation at $Re \sim 5\ 10^4$ by Ishihara et al. [2]. The simulation was performed on the Earth Simulator using a 4096^3 grid requiring 7.2 TBytes of memory to achieve a sustained speed of 16.4 TFlop/s.

The Large Eddy Simulation (LES) approach, where large scales of motion are resolved and small scales are either explicitly or implicitly modelled [3, 4, 5], provides a promising alternative to DNS. The computational requirements governed by the need to resolve the inertial range of the energy cascade are still high, however application of very-high order spatial and temporal discretisation schemes on modern HPC clusters make it possible to tackle high Reynolds number problems using LES. Efficient and scalable computational tools are a cornerstone of successful modelling of turbulent flow. In the present work, recent advancements in CFD modelling of shock physics, instabilities, transition and turbulence are presented along with the associated computing requirements. The work has been carried out using a High Resolution Computing (HIRECOM) suite which incorporates a number of single phase, multi-phase and multi-component compressible and incompressible solvers based on high-resolution schemes up to 9th-order accurate, which have been implemented in the framework of Implicit Large Eddy Simulation (ILES); see, for example, [6, 7, 8].

In unsteady turbulent flows the time step of the computation is limited by the flow physics rather than by the limitations of the stability of the numerical scheme employed, therefore explicit solvers are often preferred. One of the advantages of explicit solvers is that they can be easily parallelised based on the classical domain decomposition approach. Data exchange between the blocks and the implementation of boundary conditions are accomplished through the layer of halo cells. The application of very-high order schemes in this context leads to an increased depth of the halo required. The most accurate spatial discretisation implemented in HIRECOM at present is the 9th-order WENO (see, for example, [9]) which requires 5 halo cells on each side of the computational domain. The schematic of the 2D slice is shown in Figure 1a. The parallelisation is based on the Message Passing Interface (MPI) specification. Data exchanged between processes is stored in halo cells. For an N^3 points per core, the total number of boundary cells which should be sent/received by each process utilising a 9th-order scheme is ~$30N^2$. Therefore, it is necessary to consider the minimum number of points per core required in order to avoid the boundary exchange becoming the bottleneck. Numerical tests indicate that a distribution of ~30^3 cells per processor leads to acceptable results for most parallel architectures. Note that for this number of cells per core additional memory requirements introduced by halo cells are negligible.



a) b)

Fig. 1: *a) Computational domain(red) and halo cells for the 9th-order WENO solver; b) Parallel efficiency.*

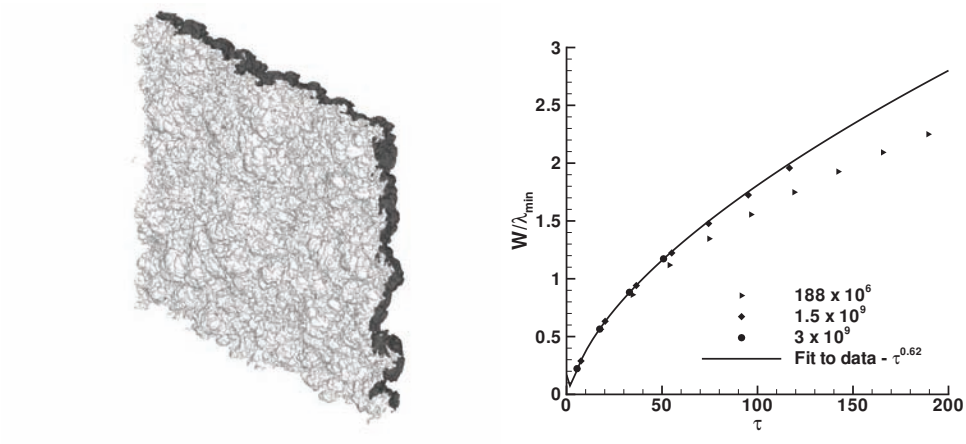
Figure 1b shows the scalability curve obtained on the HPCx facility, the UK's national high-performance computing service, for CNS3D - the compressible solver of the HIRECOM suite. A very good scalability can be demonstrated with efficiencies in excess of 80% for up to 512 processors and approximately 70% for 1024 processors (based on 100% efficiency for 96 processors). In the following sections examples from different applications in single and multiphase turbulent flows are presented.

2. SHOCK-INDUCED TURBULENT MIXING

Computations of fundamental 'building block' type flows such as shear layers, mixing instabilities and boundary layers are a cornerstone of development of modern turbulence modelling methodologies. One such flow is the Richtmyer-Meshkov instability and mixing [10, 11] across gas interfaces. This flow is of importance in the simulation of supernovae explosions, supersonic mixing processes in scramjets, and inertial confinement fusion, where mixing can promote reaction rate, or cause unwanted dilution and cooling of fuel. In the case presented here, a shock passes through an interface between two perfect gases, thus leading to Richtmyer-Meshkov instabilities, transition and turbulent mixing at late times. Very higher-order accurate schemes are employed [8] and simulations are performed on a wide range of grids ranging from 90 x 64 x 64 to 720 x 2048 x 2048, equivalent to $3\cdot 10^9$ grid points at maximum resolution.

Figure 2a shows the mixing layer at the interface at early times. Figure 2b presents the dimensionless mixing layer width W scaled by the minimum perturbation wavelength λ_{min} plotted against dimensionless time. Key to this problem is the requirement that there is sufficient separation in spatial scales between the smallest and largest flow

features - and this requires an extremely fine grid, ultimately necessitating the use of HPC facilities. Indeed, even for this 'building block' type flow configuration, convergence with respect to turbulent statistics is only observed for resolutions higher than 1440 x 1028 x 1028 - over a billion points. Results gained using HPC allowed verification of theoretical analysis of the turbulent mixing layer which is simply not possible on smaller grids run on local clusters.



a) b)

Fig. 2: *Illustrations of shock-induced turbulent mixing, a) Flow visualisation showing iso-surfaces of volume fraction, b) convergence of the dimensionless mixing layer width with respect to grid resolution.*

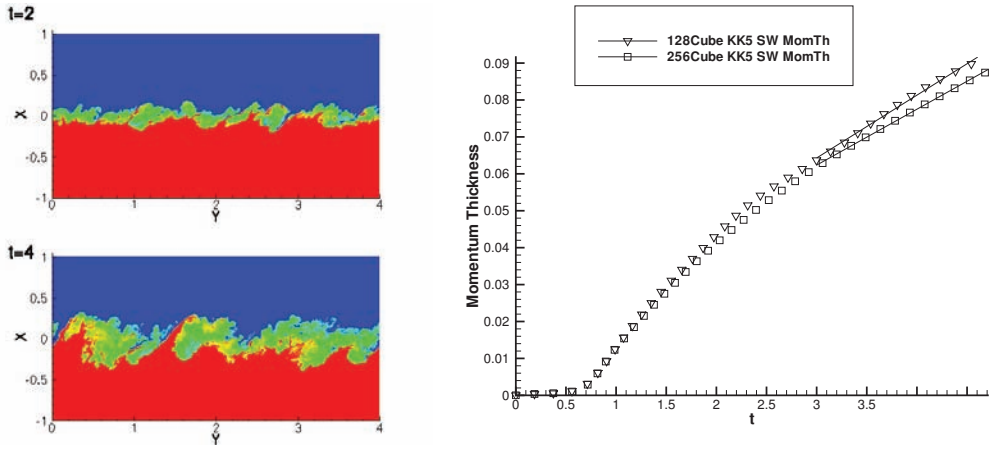
The simulations were run on Redwood Cray XT3 HPC, at AWE, Aldermaston. The facility is No 77 in the November 2008 Top 500 list and comprises 7812 AMD Opteron cores delivering ~34 TFlop/s with XT3 internal interconnect. The simulations used up to 1024 processors for up to seven days. As the code is parallelised using MPI, the efficiency of the code is dependent on the granularity, hence as long as a sufficiently large grid is employed, linear scalability of the code was observed up to the maximum number of processors employed.

3. KELVIN-HELMHOLTZ INSTABILITY AND TURBULENT MIXING

Another classical problem in turbulence which requires substantial computational resource is the Kelvin-Helmholtz instability in plane mixing layers (e.g. [12]). In the initial stages of the absolute inviscid Kelvin-Helmholtz instability, each perturbation of wavelength k grows in time proportionally to e^{kt}. Short waves grow faster than long waves and the transition is determined by the development of the shortest wave length resolved on the given grid, which makes the problem extremely sensitive to both the numerical scheme resolution and the grid employed. Following that initial transition stage, a fully developed self-similar turbulent mixing layer develops, which can be characterised by the momentum thickness δ.

ILES computations were performed on 126^3 and 256^3 grids utilising a characteristics-based scheme [13, 14] and 5th-order MUSCL reconstruction (see, for example, [15]). Figure 3 shows a snapshot of the mixing layer at $t=2$ and 4 (dimensionless) and the evolution of the momentum thickness in a 4 x 2 x 1 domain. The mixing layer development starts with the initial linearly unstable stage, characterised by the exponential growth, and undergoes transition to turbulence characterised by the linear growth of the mixing layer thickness. The 256^3 grid yields momentum thickness slope of 0.015, which is in good agreement with the experimentally observed value of 0.016. This test case illustrates the grid density required in order to accurately capture turbulence in free shear layers.

The computations were performed on Cranfield Astral HPC - an HP DL140 G3 cluster utilising a Xeon 51x0 (Woodcrest) 3GHz processors with Infiniband interconnect. The cluster comprises 856 cores and delivers 7.3TFlops Rmax on Linpack tests. Mixing layer computations on 256^3 grid required ~890 CPU hours using 128 cores and 3.6Gb RAM.

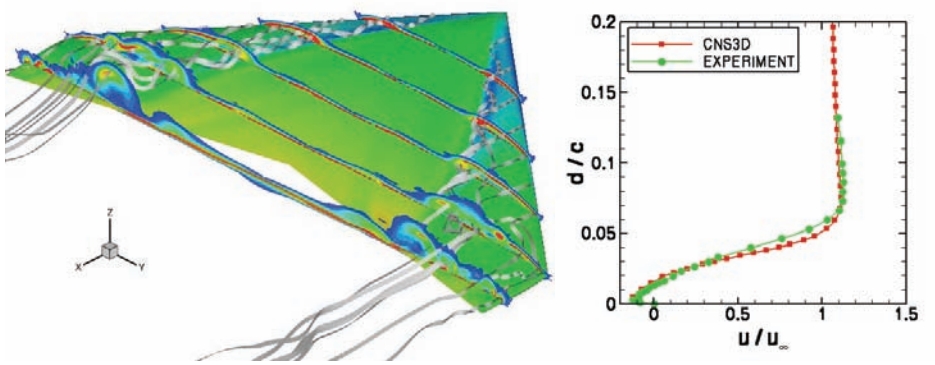


a) b)

Fig. 3: *Example of Kelvin-Helmholtz problem, a) contours of the passive scalar at t=2 and t=4 in the mid-plane, b) mixing thickness layer development with respect to grid resolution*

3. SWEPT WING SIMULATIONS

Swept and delta wings can be found in all modern aircraft travelling at transonic or supersonic speeds. At present, there are no theoretical models capable of predicting the characteristic leading edge separation with any degree of certainty, nor can the nature of the leading edge vortex breakdown observed in a swept wing flow be convincingly explained. The complexity of the flow field makes it also extremely challenging to predict with any numerical method and large computational resources are required.



a) b)

Fig. 4: *a) Instantaneous streamlines, slices of iso-vorticity contours and pressure coefficient distribution on the suction side of the wing b) Comparisons of averaged streamwise velocity at 90% half-span and 50% local chord.*

The flow around a swept wing at an angle of attack of 9^o has been simulated with the present ILES approach using high-resolution methods for a Reynolds number of 210,000 and a near incompressible Mach number of 0.3 [16]. The computational grid comprises 12.7M points yielding satisfactory near-wall accuracy with y^+ values ranging from 1 to 5. Furthermore, third-order accurate schemes (strictly speaking in the 1D context) was obtained using a modified MUSCL reconstruction method [14] and a third-order Runge-Kutta scheme with extended stability region [6]. The general physics of this problem has been captured in the ILES simulation as can be illustrated by the instantaneous flow visualisation shown in Figure 4a. The flow topology reveals transition and separation along the leading edge of the swept wing, with a subsequent formation of the characteristic vortex roll-up and vortex breakdown near the wing tips. The time-averaged velocity profiles in the fully separated, turbulent region near the

wing tip (90% half-span and 50% local chord), see Figure 4b, provide proof for the high quality of the ILES simulation. Here, data from the ILES computations is compared against experiments conducted at Manchester University (private communication with S. Zhang and J. T. Turner, Manchester University, UK, 2006). The velocity obtained by ILES is nearly identical to the experiment and the high-resolution ILES method is able to predict the correct shape and magnitude of the Reynolds stresses.

The simulations were carried out on the HPCx facility, the UK's national high-performance computing service, consisting of 160 IBM eServer 575 nodes with a total of 2560 (1.5GHz) processors.

ACKNOWLEDGEMENTS

The authors would like to thank the Engineering and Physical Sciences Research Council for the allocation of the supercomputing resources through the grants `Supercomputing Resources and Support for the UK Applied Aerodynamics Consortium' (EP/F005954/1) and `Proposal for HPCx resources and support for the LESUK3 Consortium' (EP/D053994/1), as well as the financial support through the following grants: MSTTAR Defence Aerospace Research Partnership (DARP) (GR/S27436/01), supported by EPSRC, MOD, DTI, BAE SYSTEMS, Rolls-Royce and QinetiQ; and EPSRC, MoD and AWE through the EPSRC(EP/C515153)-JGS(No. 971) project.

REFERENCES

1. S.B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.
2. T. Ishihara, T. Gotoh, and Y.Kaneda. Study of high reynolds number isotropic turbulence by direct numerical simulation. *Annu. Rev. Fluid Mech.*, 41:165-180, 2009.
3. P. Sagaut. *Large-eddy simulation for incompressible flows - An introduction*. Springer-Verlag, 2005.
4. F. F. Grinstein, L.G. Margolin, and W. J. Rider, editors. *Implicit Large Eddy Simulation*. CUP, 2006.
5. D. Drikakis. Advances in turbulent flow computations using high-resolution methods. *Progress in Aerospace Science*, 39:405-424, 2003.
6. D. Drikakis and W. J. Rider. *High-Resolution Methods for Incompressible and Low-Speed Flows*. Springer, 2004.
7. E. Shapiro and D. Drikakis. Artificial compressibility, characteristics-based schemes for variable density, incompressible, multi-species flows. Part I. Derivation of different formulations and constant density limit. *J. Comut. Phys.*, 210(2):584-607, 2005.
8. B. Thornber, A. Mosedale, D. Drikakis, D. Youngs, and Williams R. An improved reconstruction method for compressible flows with low Mach number features. *J. Comput. Phys.*, 227:4873-4894, 2008.
9. D.S. Balsara and C.-W. Shu. Monotonicity preserving weighted essentially non-oscillatory schemes with increasingly high order of accuracy. *J. Comput. Phys.*, 160:405-452, 2000.
10. R.D. Richtmyer. Taylor instability in shock acceleration of compressible fluids. *Comm. Pure Appl. Math.*, 13:297-319, 1960.
11. E.E. Meshkov. Instability of the interface of two gases accelerated by a shock wave. *I. Fluid Dyn.*, 43(5):101-104, 1969.
12. P. G. Drazin and W. H. Reid. *Hydrodynamic Stability*. CUP, 2004.
13. A. Eberle. Characteristic flux averaging approach to the solution of euler's equations. Technical report, VKI Lecture Series, 1987.
14. J. Zóltak and D. Drikakis. Hybrid upwind methods for the simulation of unsteady shock-wave diffraction over a cylinder. *Computer Methods in Applied Mechanics and Engineering*, 162:165-185, 1998.
15. K.H. Kim and C. Kim. Accurate, efficient and monotonic numerical methods for multi-dimensional compressible flows part ii: Multi-dimensional limiting process. *J. Comput. Phys.*, 208:570-615, 2005.
16. M. Hahn and D. Drikakis. Implicit Large-Eddy Simulation of Swept-Wing Flow Using High-Resolution Methods. *AIAA Journal*, 47(3):618-630, March 2009.

High Performance Computational Engineering: Putting the E Back in CSE

Dimitri J. Mavriplis*

**Department of Mechanical Engineering*
University of Wyoming, Laramie, WY 82071
(e-mail: mavripl@uwyo.edu)

Abstract: Over the last decade, advocacy for high performance computing has increasingly been taken up by the science community with the argument that computational methods are becoming a third pillar of scientific discovery alongside theory and experiment. Computational engineering, on the other hand, has continually been relegated to a set of mature software tools which run on commodity hardware, with the notion that engineering problems are not complex enough to warrant the deployment of state-of-the-art hardware on such a vast scale. We argue that engineering practices can benefit equally from an aggressive program in high performance computational methods, and that these problems are at least as important as science problems, particularly with regards to national competitiveness objectives. Computational engineering problems differ in various ways from computational science problems, and specific techniques for addressing these differences on large scale parallel hardware need to be developed. Traditionally, computational fluid dynamics, and more specifically, computational aerodynamics have been principal drivers for advances in high-performance computational engineering. We argue that these fields are best suited to resume this leadership role in computational-based engineering developments, and describe a set of Grand Challenge problems which can be used to motivate and demonstrate the benefits of high-performance computing for aerospace engineering problems.

Keywords: high-performance, computing, aerospace, engineering

1. INTRODUCTION

By all current accounts, the path towards increasingly high performance computing (HPC) for applications running at sustained petaflops involves radically higher levels of parallelism. This is evident in the rapid growth in the number of cores of current and planned leadership class machines due to the emergence of multicore architectures, as well as the appearance of GPUs and Cell processors on the scientific computing stage, which achieve high performance through extensive multithreading. While much of the interest and advances in high performance computing have been achieved in various science application domains, advances in HPC for engineering application domains has received less attention over the last decade. This has been noted in a recent NSF report entitled Simulation Based Engineering Science [1], where the need advances in various important engineering application fields has been outlined, and the special needs of engineering problems has been discussed. More recently, a Computational-Based Engineering Summit was held under the auspices of the NSF, Sandia National Laboratory, the National Academy of Engineering and the Council on Competitiveness to examine these same issues [2].

Engineering application problems differ from science applications in various ways which complicate the porting of these applications to current and future planned petaflops architectures.

Engineering problems are most often concerned with a small number of specific objectives, which are needed to evaluate product performance or guide design decisions. Furthermore, the accuracy required for useful simulations in the design process is highly variable, where sometimes crude ball-park accuracy is sufficient, while at other times high accuracy is required. In most cases however, good accuracy in the relevant objective does not require fully resolving all physical phenomena present in the governing physics, but only those that have an important impact on the important objectives. These characteristics of engineering simulations often mean that extensive spatial resolution (for example billion plus grid cell calculations) is not necessary for many problems. However, these problems remain extremely computationally intensive due to the need to take large numbers of small time steps, the cost of solving stiff non-linear systems at each time step, and in the case of design optimization problems, the cost of performing large numbers of optimization steps. Since most current-day applications rely on spatial domain decomposition for achieving parallelism, most engineering simulations have lagged at exploiting the levels of parallelism needed for achieving petaflops simulations on the most recent hardware.

Conversely, in cases where high spatial accuracy is required, the complex geometries most often associated with engineering problems has made the generation of highly resolved meshes (billion plus cells) impractical, since specialized (mostly commercial) grid generation and geometry definition software is required, which most often runs in serial. Additionally, these large mesh data-sets must be loaded onto the parallel machine at startup, and retrieved along with the computed solution at each (or selected) time step of the simulation, for time dependent dynamic mesh problems, constituting an I/O bottleneck on current petascale hardware.

A less specific but equally important obstacle to increased investments in HPC for simulation-based engineering projects is a notion that engineering problems do not require ever increasing amounts of computational power. While it is true that capabilities exist that are used successfully in every-day engineering calculations, radical advances in simulation capability are possible through the coupling of increased computational power with more capable algorithms. Historically, simulation-based engineering capabilities have been developed and demonstrated using public funding. As the potential benefits of these technologies becomes apparent, industry has generally been very eager to adopt these technologies and integrate them into their product development cycle, often making the large investments required in terms of hardware, software and personnel. However, once the capabilities become integrated, industrial emphasis more often turns to the goal of reducing the cost of this fixed simulation capability (generally through migration to newly available cheaper hardware) rather than seeking increases in simulation capabilities at fixed cost (i.e., by continually acquiring the latest available high end hardware). For example, the development of computational aerodynamics in the aerospace community was characterized by a continual drive to higher fidelity (and more accurate) methods from the 1970's to the 1990's, beginning with panel methods, proceeding to linearized and nonlinear potential flow methods, inviscid flow (Euler) methods, and culminating with Reynolds-averaged Navier-Stokes (RANS) methods in the 1990's. Throughout this development period, effective use of these methods required investment in the most capable high-end computing hardware available at the time, an investment industry was more than willing to shoulder in return for the improved simulation capability. In the early 1990's, it would not be uncommon for larger corporations to house HPC hardware on a par with most of the leading national labs, for either aerodynamics in the aerospace industry, or crash simulations in the automobile industry.

However, the last decade has seen a stagnation of the capabilities used in aerodynamic simulation in the aerospace industry, with RANS simulations having become the high-fidelity simulation method of choice, and emphasis being directed towards reduction in the cost of RANS simulations through migration towards more cost-effective but no longer leading-edge computational hardware.

The science community has been very effective in demonstrating scientific discoveries that can be made possible through advances in HPC. Similarly, the engineering community should strive to demonstrate the benefit of HPC advances in this field. The formulation of Grand Challenge problems can be used for this purpose. The intent in defining these Grand Challenge problems is to set long-term goals for driving the essential developments that will be required to achieve these and other types of advances in simulation capabilities, and to illustrate the new frontiers that such capabilities would enable. In the final paper, we will firstly outline the specific ways in which computational engineering problems differ from computational science problems, and will formulate and discuss various Grand Challenge aerospace engineering problems which can be used to both demonstrate the benefits of HPC in aerospace engineering, and to drive advances in the state-of-the-art.

REFERENCES

1. Simulation Based Engineering Science: *Revolutionizing engineering science through simulation. NSF Report*, May 2006. Report of the NSF Blue Ribbon Panel on Simulation Based Engineering Science.

2. *Transforming Engineering through Computational simulation*. http://www.sandia.gov/tecs/TECSsummit.html.

3. Mavriplis, D. J., Darmofal, D., Keyes, D., and Turner, M.: *Petaflops Opportunities for the NASA Fundamental Aeronautics Program*, AIAA Paper 2007-4084, Presented at the 18th AIAA Computational Fluid Dynamics Conference, Miami FL, June 2007.

Building-Cube Method: A Block-Structured Cartesian Grid Approach for Near-Future Peta-Flops Computers

Kazuhiro Nakahashi*, Shun Takahashi, Takashi Ishida*****

** Department of Aerospace Engineering, Tohoku University, Sendai 980-8579, JAPAN*
(Tel: +81-22-795-6978; e-mail:naka@ad.mech.tohoku.ac.jp)
***Postdoctoral fellow, Department of Aerospace Engineering, Tohoku University, JAPAN*
(e-mail: takahasi@ad.mech.tohoku.ac.jp)
****Ph.D. candidate, Department of Aerospace Engineering, Tohoku University, JAPAN*
(e-mail: ishida@ad.mech.tohoku.ac.jp)

Abstract: Currently CFD has become an indispensable tool for analyzing and designing aircrafts. Wind tunnel testing, however, is still the central player for aircraft developments and CFD plays a subordinate part. In this article, demands for next-generation CFD are described with an expectation of near future PetaFlops computers. Then, Cartesian grid approach, as a promising candidate for next-generation CFD, is discussed by comparing it with the current unstructured grid CFD. It is concluded that the simplicity of Cartesian mesh CFD from the mesh generation to the post processing will be a big advantage in the days of PetaFlops computers.

Keywords: Cartesian grid, Large-scale computation

1. WILL CFD TAKE OVER WIND TUNNELS?

More than 20 years ago, I heard an elderly physicist in fluid dynamics say that it was as if CFD were just surging in. Other scientists of the day said that with the development of CFD, wind tunnels would eventually become redundant.

Impressive progress in CFD has been made during the last three decades. In the early stage, one of the main targets of CFD for aeronautical fields was to compute flow around airfoils and wings accurately and quickly. Body-fitted-coordinate grids, commonly known as structured grids, were used in those days (Fig. 1).

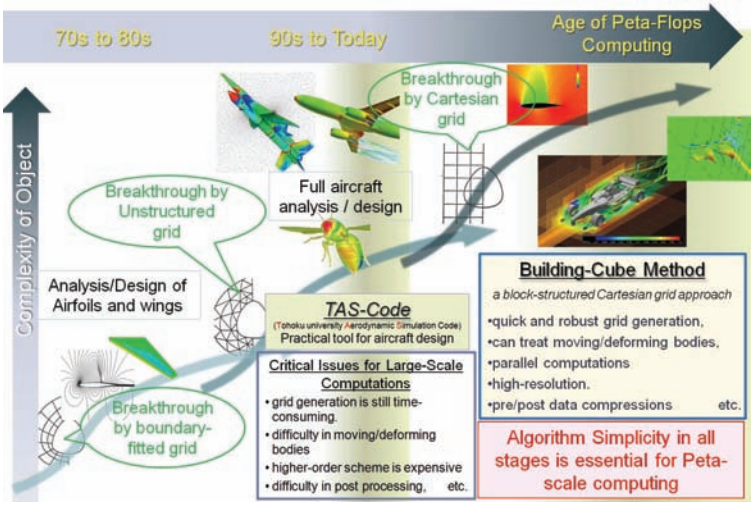


Fig.1: *Progress of Aeronautical CFD*

From the late eighty's, the target was moved to analyzing full aircraft configurations [1]. This spawned a surge of activities in the area of unstructured grid CFDs. Unstructured grids provide considerable flexibility in tackling complex geometries [2]. CFD has become an indispensable tool for analyzing and designing aircrafts.

So, is CFD taking over the wind tunnels as predicted twenty years ago?

Today, Reynolds-averaged Navier-Stokes (RANS) computations can accurately predict lift and drag coefficients of a full aircraft configuration. It is, however, still quantitatively not reliable for high-alpha conditions where flow separates. Boundary layer transition is another cause of inaccuracy. These are mainly due to the incompleteness of physical models used in RANS simulations. Large Eddy Simulation (LES) and Direct Numerical Simulation (DNS) are expected to reduce the physical model dependencies. But we have to wait for the further progress of computers for the use of those large-scale computations in engineering purposes.

For the time being, the wind tunnel is the central player and CFD plays a subordinate part in aircraft developments.

2. RAPID PROGRESS OF COMPUTERS

The past CFD progress has been highly supported by the improvements of computer performance. The latest Top500 Supercomputers Sites [3] tell us that the performance improvement of computers has reached a factor of 1000 in the last 10 years as shown in Fig. 2. Increase in the number of processors in a system in addition to the degree of integration contributes to this rapid progress.

With a simple extrapolation of Fig. 2, we can expect to use PetaFlops computers soon. This will accelerate the use of 3D RANS computations for the aerodynamic analysis and design of entire airplanes. DNS which does not use any physical models may also be used for engineering analysis of wings. In the not very far future, CFD could take over wind tunnels.

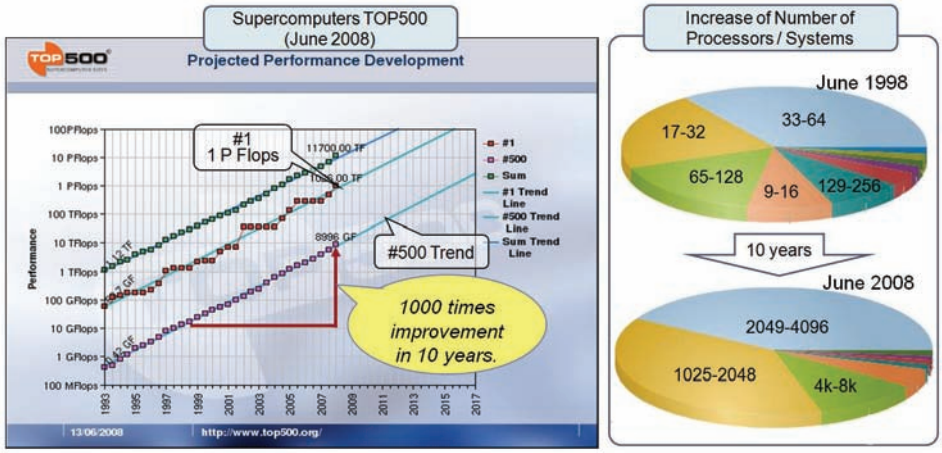


Fig.2: *Performance development in Top500 Super-computers [3]*.

3. DEMANDS FOR NEXT-GENERATION CFD

So, is it enough for us as CFD researchers to just wait for the progress of computers? Probably it is not. Let's consider demands for next-generation CFD on PetaFlops computers.

1. *Easy and quick grid generation around complex geometries,*
2. *Easy adaptation of local resolution to local flow characteristic length,*
3. *Easy implementation of spatially higher-order schemes,*
4. *Easy massively-parallel computations,*
5. *Easy post processing for huge data output,*
6. *Algorithm simplicity for software maintenance and update.*

Unstructured grid CFD is a qualified candidate for the demands 1 and 2 as compared to structured grid CFD. However, an implementation of higher-order schemes on unstructured grids is not easy. Post processing of huge data output may also become another bottleneck due to irregularity of the data structure.

Recently, studies of Cartesian grid method were renewed in the CFD community, because of the several advantages such as rapid grid generation, easy higher-order extension, and simple data structure for easy post processing. This is another candidate for the next-generation CFD.

4. BUILDING-CUBE METHOD

A drawback of uniform Cartesian grid is the difficulty of changing the mesh size locally. This is critical, especially for airfoil/wing computations, where an extremely large difference in characteristic flow lengths exists between boundary layer regions and far fields. Accurate representation of curved boundaries by Cartesian meshes is another issue.

A variant of the Cartesian grid method is to use the adaptive mesh refinement in space and cut cells or the immersed boundary method on the wall boundaries. However, introduction of irregular subdivisions and cells into Cartesian grids complicate the algorithm for higher-order schemes. The advantages of the Cartesian mesh over the unstructured grid, such as simplicity and less memory requirement, disappear.

The present author proposes a Cartesian grid based approach, named Building-Cube method [4]. Basic strategies employed here are; (a) zoning of a flow field by cubes of various sizes to adapt the mesh size to local flow characteristic length, (b) uniform Cartesian mesh in each cube for easy implementation of higher-order schemes, (c) same grid size in all cubes for easy parallel computations, (d) staircase representation of wall boundaries for algorithm simplicity.

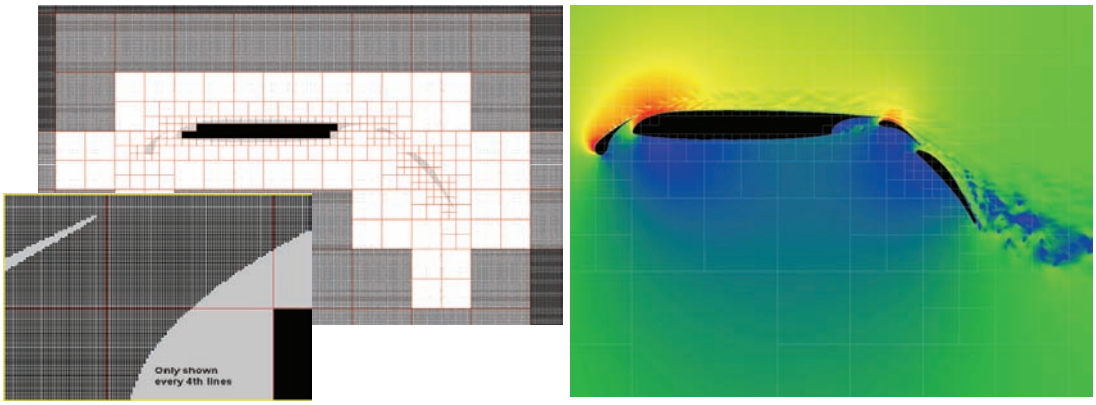


Fig. 3: *BCM mesh (481 cubes with 256x256 Cartesian mesh in each cube) and the instantaneous Mach distributions for a four-element airfoil; Re=2.83x10^6, M_∞=0.201, $\alpha = 8.16°$*

It is similar to a block-structured uniform Cartesian mesh approach [5], but unifying the block shape to a cube simplifies the mesh generation [6] and the domain decomposition of a computational field around complex geometry. Equality of computational cost among all cubes significantly simplifies the massively parallel computations as shown in Fig. 4. It also enables us to introduce data compression techniques for pre and post processing of huge data [4].

A staircase representation of curved wall boundaries requires very small grid spacing to keep the geometrical accuracy as shown in Fig. 3. But the flexibility of geometrical treatments obtained by it will be a strong advantage for complex geometries and their shape optimizations. Although computation using high-density Cartesian mesh is still far from practical use because of the computational time, it will be resolved with a progress of computers.

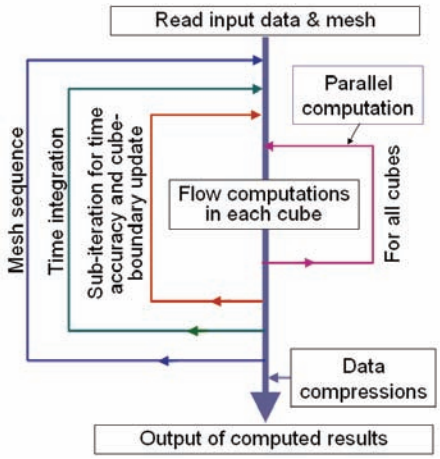


Fig.4: *Overall flow-solution procedure*

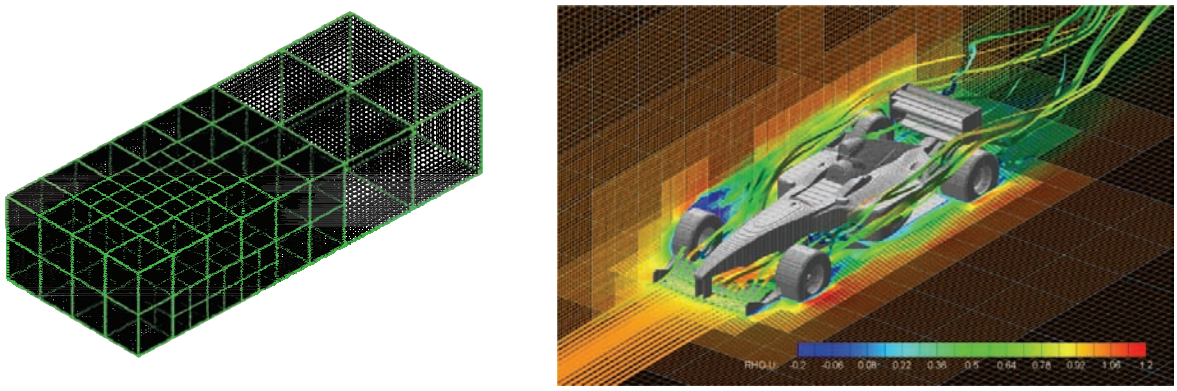


Fig. 5: BCM mesh (5930 Cubes with 32x32x32 mesh in each cube, totally about 200 million cells) and the time-averaged velocity distribution of the incompressible flow computation [7].

5. SIMPLICITY IS ESSENTIAL FOR NEXT-GERATION CFD

CFD, using a high-density Cartesian mesh, is still limited in its application due to the computational cost. The predictions about Cartesian mesh CFD and computer progress in this paper may be too optimistic. However, it is probably correct to say that the simplicity of the algorithm from grid generation to post processing of Cartesian mesh CFD will be a big advantage in the days of PetaFlops computers.

REFERENCES

1. Jameson, A. and Baker, T. J. (1987). Improvements to the Aircraft Euler Method, *AIAA Paper 1987-452*.
2. Nakahashi, K., Ito, Y. and Togashi, F. (2003). Some Challenges of Realistic Flow Simulations by Unstructured Grid CFD, *Int. J. for Numerical Methods in Fluids,* Vol.43, pp.769-783.
3. Top500 Supercomputers Sites (2008). http://www.top500.org/.
4. Nakahashi, K. (2005). High-Density Mesh Flow Computations with Pre-/Post-Data Compressions, *AIAA 2005-4876, Proc. AIAA 17th CFD Conf.*.
5. Meakin, R. L. and Wissink, A. M. (1999). Unsteady Aerodynamic Simulation of Static and Moving Bodies Using Scalable Computers, *AIAA-99-3302, Proc. AIAA 14th CFD Conf.*
6. Ishida, T., Takahashi, S., Nakahashi, K. (2008). Efficient and Robust Cartesian Mesh Generation for Building-Cube Method, *J. of Computational Science and Technology*, Vol.2, No.4, pp.435-446.
7. Takahashi, S., Ishida, T., Nakahashi, K., Kobayashi, H., Okabe, K., Shimomura, Y., Soga, T., and Musa, A. (2008). Large Scaled Computation of Incompressible Flows on Cartesian Mesh Using a Vector-Parallel Supercomputer, *Int. Conf. on Parallel Computational Fluid Dynamics*, Lyon, France.

CFD APPLICATIONS FOR NASA'S SPACE EXPLORATION MISSION

CFD—Mature Technology for Space Exploration Mission Support?

Rupak Biswas*, Dochan Kwak, Eugene L. Tu*****

** NASA Advanced Supercomputing Division, NASA Ames Research Center, Moffett Field, CA 94035, USA*
(Tel: 650-604-4411; Rupak.Biswas@nasa.gov)
*** NASA Advanced Supercomputing Division, NASA Ames Research Center, Moffett Field, CA 94035, USA*
(Tel: 650-604-6743; Dochan.Kwak@nasa.gov)
**** Exploration Technology Directorate, NASA Ames Research Center, Moffett Field, CA 94035, USA*
(Tel: 650-604-4486; Eugene.L.Tu@nasa.gov)

Abstract: With the maturation of CFD technology and staggering advances in supercomputing capability, CFD applications are becoming increasingly vital and cost effective for the design and analysis of spaceflight vehicles and related launch operations. We present a brief summary of progress in CFD technology as well as discuss some pacing challenges that must be overcome, especially for space exploration applications.

Keywords: Spaceflight, launch environment, numerical validation, predictive capability, supercomputing.

1. INTRODUCTION

Supporting NASA's exploration mission using CFD has been challenging due to the wide spectrum of speed regimes, the complexity of geometries, and the fundamentally diverse flow physics that are involved. Early design practices for developing space flight vehicles and operational procedures relied heavily on experimental, test, and flight data. As CFD technology continues to mature along with exponential advances in supercomputing capability, CFD applications become increasingly important and cost-effective for the design and analysis of flight vehicles and related launch operations. Just as it was in aviation and aircraft design, the primary question now is: How mature is CFD in supporting NASA's exploration mission? Since such applications involve inherently large numbers of grid points and massive computing resources, the efficient parallel implementation of simulation tools adds one more dimension to traditional CFD technology. In this paper, current capabilities and maturity will be demonstrated through recent examples followed by the authors' perspective on what advances must be made to make CFD more useful to mission development and operations.

2. EVOLUTION OF CFD CAPBILITIES IN AEROSPACE ENGINEERING

There are a vast number of cases where the CFD approach has made significant impact on aerospace engineering. In this section, a short summary of progress in CFD applications is given from a historical perspective. The examples included below illustrate the level of complexity researchers have encountered in fluid engineering as modeling and fidelity requirements have increased and CFD technology has evolved.

2.1 Aeronautics

Application of CFD tools to engineering problems became realistic in the early 1970s, as high-speed scientific computers were becoming increasingly available. Algorithms that had been developed earlier were further extended, utilizing then high-speed processors. However, computational speeds were still too limited to produce Navier-Stokes (N-S) solutions for problems with complicated geometry. Starting with simplified formulations, numerous successful methods and tools were developed and applied to real-world design problems, with the most notable success in commercial airplane designs. Processor speed increased from a fraction of a gigaflop in the early '80s to tens of gigaflops in the '90s. With this increased computer speed, full N-S solutions for a complete aircraft configuration became possible. Since then, the goal of CFD simulation has advanced further to tackle drag prediction, unsteady N-S computations, and multidisciplinary optimizations for full aircraft geometry.

2.2 Aerodynamics of Spaceflight Vehicle

The CFD technology and experience gained from aeronautics can be extended to aerodynamic analysis of space flight vehicles. However, flow simulations involving these vehicles, for example the Space Shuttle and other expendable launch vehicles, are significantly more challenging compared to aerodynamically shaped vehicles like commercial aircraft. Computational challenges include complex geometry such as protuberances, multiple bodies in relative motion as encountered in stage separation, boundary layer plume interaction, and transient flow through variable speed regimes. Related physical modeling involves non-equilibrium turbulence, chemistry, blast wave, and aero-acoustics. Validation is also very difficult due to insufficient test or flight data. From an engineering point of view, CFD tools can still offer alternatives to an entirely empirical approach and have become indispensable in developing exploration vehicles and operations. Often, CFD applications to mission support relied on "best practices" and some of these results will be presented to illustrate that CFD adds a new dimension to the "test-fail-fix" procedure of the past.

2.3 Ground Operations and Launch Environment

Ground operations involve safety and risk assessment of the Vehicle Assembly Building (VAB) at NASA Kennedy Space Center. For example, if an accidental ignition of a stored Solid Rocket Booster (SRB) were to occur and subsequently lead to propagation of pressure waves and toxic gas to nearby personnel and structures, it is of critical importance to estimate resulting damages and risks. This problem can only be addressed through computational modeling. CFD tools provide the basis for developing a reliable and sophisticated modeling procedure. This simulation requires unconventional modeling of the ignition and propagation of fire. Due to the large spectrum of spatial and temporal scales, computing time and memory requirements push the boundaries of current supercomputing capabilities.

Another area where supercomputing enables the quantification of flow-related issues is in the launch environment. Simulations that have been used to generate data for design decisions have included the flame trench, propagation of the ignition over pressure (IOP) phenomenon, and aero-acoustic loads generated from the unsteady/transient exhaust plumes during launch. High-fidelity unsteady/transient flow simulation is necessary, and inclusion of multi-phase flow modeling is required to account for the water suppression system, activated to suppress the IOP waves. Previously, impact of acoustic waves on the vehicle has been estimated using empirical correlations. However, high-fidelity unsteady CFD can be utilized to more accurately model the entire launch environment.

2.4 EDL and Aerothermodynamics

Entry-Descent-Landing (EDL) poses enormous challenges to CFD simulations, especially in the hypersonic flow regime. NASA's latest aerodynamic design of entry vehicles was based on Viking technology from 1976. With current supercomputing capability, it is possible to make revolutionary advances in entry technology that is required for large mass landing on planets and the Moon, as well as some Earth entry conditions. The geometry of entry vehicles is relatively simple; however, the physics related to EDL requires major advances in CFD technology, especially the modeling of high-energy physics, physics-based transition, and high-fidelity turbulence. Such multi-physics flow simulations require high-accuracy numerical procedures and large computing resources.

2.5 Propulsion System

Numerical methods and boundary condition procedures have advanced since the 1980s to handle complex rotor-stator interaction problems encountered in turbine engines. Yet the computational requirements for modeling and simulating even a limited number of a rotor-stator rows are so huge that it took many months to complete just one calculation involving a single stage of a rotor-stator—making it impossible to apply CFD simulation to a turbine or compressor design. Primarily due to the computer hardware speedup, it became possible to analyze multi-stage turbine flow in the 1990s and 2000s. Yet despite these advances, the impact on jet engine design is still limited to the component level. This is partially due to inability of the current physics model in predicting highly accurate flow quantities.

Rocket propulsion CFD has, in general, lagged behind aircraft engine applications. The complexity of the flow physics and hardware geometry involved in modeling rocket engines has delayed the application of CFD to this area. One of the more significant applications of CFD simulation to rocket engine analysis began in the early

‘80s, when NASA carried out a series of upgrades to the Space Shuttle Main Engine (SSME), originally developed in the ‘70s. One such effort involved the powerhead redesign, which resulted in successful development of an upgraded flight engine in the mid-90s. The liquid-fueled engine includes complex internal flow and turbopumps. One critical role CFD played in recent years is related to the Shuttle flow liner that required high-fidelity unsteady flow simulation through the SSME Low Pressure Fuel Turbopump (LPFTP) and around the flow liner located just upstream of LPFTP. Simulation results helped identify the root cause of the flow liner crack failure, and to determine flight rationale for a safe mission. These calculations would not have been possible without the combination of supercomputing capability, engineering knowledge, and advanced computational modeling tools.

2.6 Human Modeling for Spaceflight

Finally, application of CFD methods to blood flow has been of interest to biomedical researchers for many years. However, the lack of a complete analysis capability prevented it from making significant impacts on medical research and practices. For human spaceflight, biomedical performance modeling for astronauts is essential. To date, this is done via empirical correlations. However, available flight data is limited to the maximum duration of 6-month stays on the International Space Station. For longer space travel, extrapolation of the current data, either from flight or ground-based (e.g. from artificial-gravity experiments), is difficult. Some of the critical information needed includes bone and muscle loss mechanism, and impact of altered gravity on blood circulation in the brain. These can be supported by CFD along with physiological models. Supercomputing capability can provide an extra dimension to this “digital astronaut” type human space flight modeling yet to be realized.

3. CHALLENGES AND POSSIBILITIES

In general, CFD capabilities have been advanced along with computational technologies. For example, flow solver codes and software tools have been developed to the point that many daily fluid engineering problems can now be computed routinely. However, especially for space exploration applications, there remain some pacing challenges.

3.1 Acceleration of Solution Procedures

Flow simulations for space exploration applications generally involve very complex geometries, flow physics, and flight envelopes; thereby requiring substantial computing resources. Specifically, resolving unsteady phenomena is becoming increasingly important in order to fully understand the fluid dynamics issues. A typical process of flow simulation, especially for high-fidelity unsteady flow, requires large amounts of both computing time and human time in problem set-up and data pre-/post-processing. A substantial reduction in overall computational time for 3-D unsteady flow simulations is required to enable unsteady CFD to become relevant to mission-critical decision making. A portion of this speedup will come from enhancements in computer hardware; however, the remainder must be contributed by advances in grid-generation procedures, flow solution algorithms, and more efficient parallel implementations.

3.2 Prediction of Physics

Currently, the predictive capability of flow simulation is generally quite limited. The accuracy of physical models is one of the major bottlenecks and needs to be further developed. Advanced fluid dynamics models including turbulence and transition prediction, chemical reaction, and cavitation physics must be included for accurate prediction of mission-related tasks. In addition, other quantities like thermal stresses and structural loads must be coupled to the fluid dynamics models to provide more realistic simulation capabilities in an inherently multi-disciplinary environment. These computations will require not only large computing resources, but massive data storage and efficient data management technologies as well.

Perhaps one of the most critical issues in physical modeling remains the turbulence model. Those routinely used in production CFD codes are based on equilibrium turbulence, i.e. an eddy viscosity model. However, as the requirement to apply CFD to engineering problems with complex flow physics increases, correct prediction of time-dependent turbulence and non-equilibrium phenomena become necessary. For example, in aerospace design, the most productive aspect of CFD applications has often been to predict relative change among design variations. Even for this trend prediction, higher-level turbulence modeling is necessary. To push the limits of

operational boundary and try bold new ideas, more predictive capabilities will be needed in the near term for complicated flows involving transient phenomena, separation, tip vortex interaction, and cavitation. To make these advances, high-fidelity computations using supercomputing resources will play a key role in CFD.

3.3 Human Resources and CFD Validation

The human resources aspect of applying CFD to exploration missions must also be considered. Even though CFD has advanced remarkably, many challenging cases require experts. Computer science can automate a substantial portion of the CFD simulation processes, thus saving much human time required to obtain solutions. However, blind application of tools without understanding the capabilities and limitations of the methods involved could lead to catastrophic engineering results. As in many other engineering and science disciplines, CFD researchers and practitioners need to understand the physics and engineering systems being simulated. More rigorous and defined processes and procedures need to be developed to validate the CFD solutions and to provide engineering error estimation and repeatability of the results. In short, the application of CFD to engineering problems needs to be as well understood as a ground-based experiment or flight test. Future experts in the application of CFD must be cultivated to think through the relevant flow physics and apply the appropriate software to the engineering problem to succeed.

Finally, the above statements are made without proofs or references, and compiled into a single graph as shown in Figure 1. This figure is thus intended to illustrate the evolution of CFD activities in support of space exploration as a function of computer hardware advances. In the full-length version of this paper, more details will be provided. The material presented here is collected from many accomplished researchers in the Exploration Technology Directorate at NASA Ames Research Center.

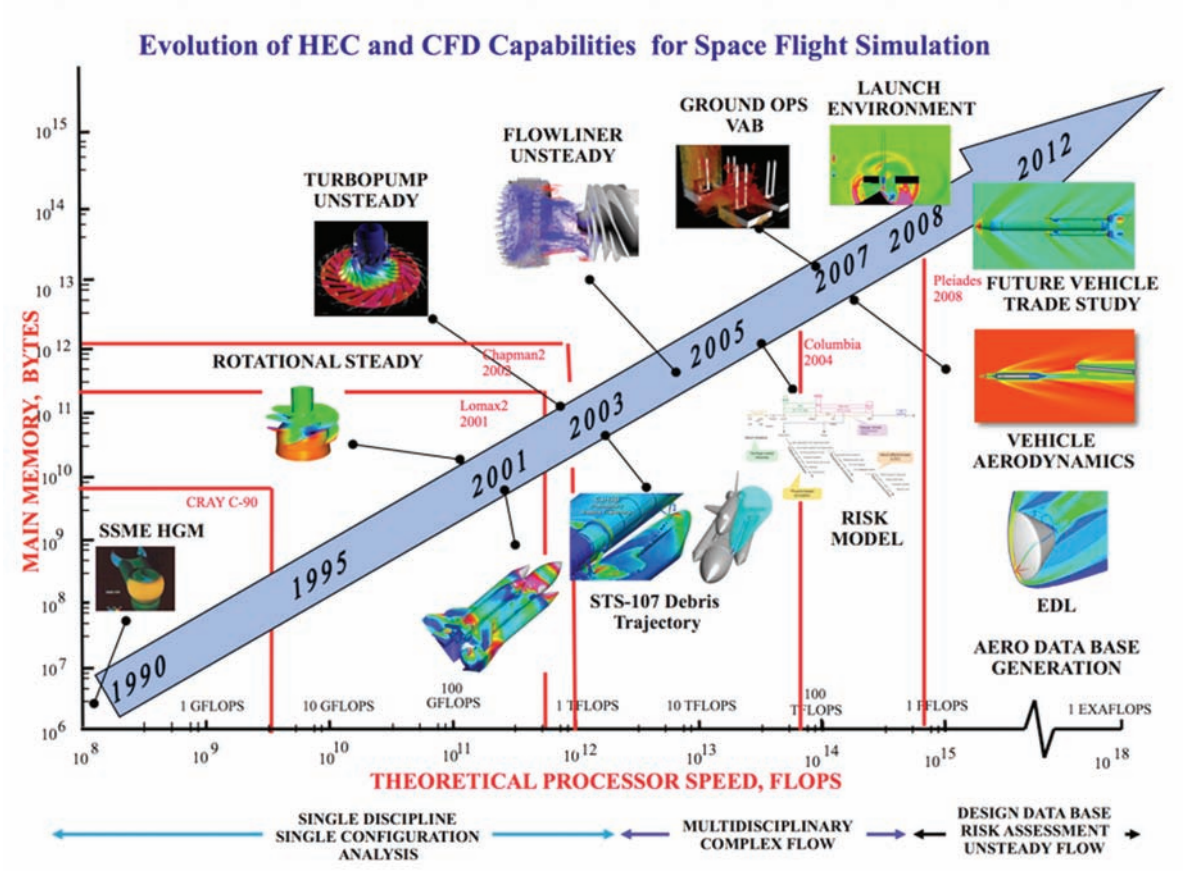


Fig. 1: *Evolution of high-end computing and examples of CFD applications to exploration mission.*

NASA's Space Operations Missions Directorate Parallel Computing Applications

Reynaldo J. Gómez III*

*NASA Johnson Space Center, Houston, TX, 77058, USA
(Tel: 281-483-6108; e-mail: reynaldo.j.gomez@nasa.gov)

Abstract: NASA's Space Shuttle and International Space Station Programs routinely use large parallel computing systems to simulate a wide range of aerodynamic, aerothermodynamic, and debris impact environments. From ground winds and launch pad ignition over pressure environments, to on-orbit aerodynamics and hypervelocity orbital debris impacts, and back down through a hypersonic, non-equilibrium, chemically reacting flowfield, parallel computing systems provide key insights into critical environments that must be understood in order to safely carry out NASA's space operations goals.

Keywords: Spaceflight, aerodynamics, aerothermodynamics, debris

1. Background

The Space Shuttle was developed prior to the availability of large parallel computing systems since the 1970's state of the art in aerodynamic prediction technology was largely based on empirical data from wind tunnels. During the 1980's parallel vector processors, coupled with advances in physical modelling of fluid flows, enabled the first simulations of the complete launch vehicle, albeit with simplified geometry and overly coarse spatial discretization. Building on these simulation tools and leveraging exponential increases in parallel computing performance during the 1990's improved the Shuttle simulations by an order of magnitude in geometric fidelity and resolution, allowing the first accurate computational predictions of the ascent aerodynamic environment.

Mounting needs to provide high fidelity loads and thermal analysis in rarefied conditions, for both high altitude atmospheric flight and on-orbit plume environments, motivated the development of a highly parallelized implementation of the Direct Simulation Monte Carlo technique. This capability was initially leveraged to simulate Shuttle/Mir rendezvous and docking loads generated by the Shuttle Reaction Control System plumes impinging on the Mir space station. Later applications included both plume impingement and high fidelity orbital drag predictions for ISS, Space Shuttle servicing mission environments to the Hubble Space Telescope, as well as rarefied airloads estimates on Earth and interplanetary satellites.

Following the loss of STS-107, parallel computing systems were inundated with analyses to understand the cause of the damage to the Space Shuttle Orbiter Columbia and the environments that results in burn through of the wing leading edge. Efforts to understand the detailed flowfield around the External Tank and debris trajectories from foam and ice losses on the vehicle motivated the development of high fidelity CFD models of the External Tank and new debris transport tools to simulated unsteady dynamic trajectories. Non-linear structural impact models were used to simulate debris impacts on various components of the Shuttle. And numerous entry simulations were performed with various types of damage to determine the most probable damage scenario and later to develop capabilities to rapidly assess inflight damage to the Shuttle Orbiter.

The tools developed to simulate ascent, entry and internal flow environments for the Shuttle and on-orbit environments for the ISS have been used to address a wide range of issues that have been encountered during the operational life of these vehicles and have been applied to many other vehicles by academia, commercial and military applications. Despite the advances in physical modeling and parallel computing, and the contributions that have been made over the life of the Space Shuttle, several key environments cannot be accurately simulated with the available tools or test facilities. Simulations of these key environments are typically hampered by physical modeling limitations and reduction or removal of these limitations is where additional research and tools development are needed.

REFERENCES

1. Buning, P.G., Chiu, I.T., Obayashi, S., Rizk, Y.M., and Steger, J.L., "Numerical Simulation of the Integrated Space Shuttle Vehicle in Ascent," AIAA-88-4359-CP, August 1988.
2. Slotnick, J.P., Kandula, M., and Buning, P.G.,"Navier-Stokes Simulation of the Space Shuttle Launch Vehicle Flight Transonic Flow field Using a Large Scale Chimera Grid System," AIAA-94-1860, AIAA 12th Applied Aerodynamics Conference, Colorado Springs, CO, June 1994.
3. Gomez, R., Vicker, D., Rogers, S.E., Aftosmis, M.J., Chan, W.M., Meakin, R., and Murman, S.M., (2004). "STS-107 Investigation Ascent CFD Support," AIAA Paper 2004-2226.
4. Murman, S.M., Aftosmis, M.J., and Rogers, S.E., (2005) "Characterization of Space Shuttle Ascent Debris Aerodynamics Using CFD Methods," AIAA Paper 2005-1223.
5. Brown, J. D., Bogdanoff, D. W., Yates, L. A., Wilder, M. C., and Murman, S.M., (2006) "Complex-Trajectory Aerodynamics Data for Code Validation from a New Free-Flight Facility," AIAA Paper 2006-0662.
6. Palmer, G. E., Pulsonetti, M. V., Wood W. A., Alter, S., Gnoffe, P. A. and Tang, C. (2008). "CFD Assessment of Thermal Protection System Tile Damage Experienced During STS-118," AIAA-2008-424
7. Rogers, S. E. (2007). "Hemisphere Ice Balls Debris Transport Analysis," NAS Technical Report, NAS-07-004.
8. Murman, S. M. (2008) "Dynamic Stability Analysis using CFD Methods," in *Experimental Determination of Dynamic Stability Parameters, Von Karman Institute for Fluid Dynamics*.
9. Fahrenthold, E. (2004). "Simulation of Foam Impact Effects on Components of the Space Shuttle Thermal Protection System," AIAA-2004-940.
10. Park, Y.K, Fahrenthold, E.,(2006) "Simulation of Hypervelocity Impact Effects on Reinforced Carbon-Carbon" *Journal of Spacecraft and Rockets,* Vol. 43, No. 1.

High Performance Computing Applications for Development of the Orion Aeroscience Flight Databases

Joseph Olejniczak
NASA Ames Research Center
M/S 230-3
Moffett Field, CA 94035

A multi-Center NASA team is responsible for producing the complete aerodynamic and aerothermodynamic databases for the Orion Crew Module (CM) and Launch Abort System (LAS) covering the range of all possible operating conditions. The databases will be developed using both computational tools and wind tunnel testing. The development of the databases will require thousands of high fidelity numerical solutions that model the flow field around the CM and LAS for all flight regimes. The databases will be used to both design and operate the vehicle.

Accurate aerodynamic data such as lift, drag, pitching moment, and dynamic stability derivatives are required to design the flight control system and ensure that the pinpoint landing requirement can be met. The aerodynamic database covers the entire CEV operational envelope including nominal ascent, ascent abort scenarios, on-orbit plume environments, re-entry flight from the hypersonic through subsonic regimes, and the terminal landing approach including parachute deployment.

The aerothermodynamic database covers the portion of atmospheric flight that produces significant aeroheating to the vehicle. While the ascent heating environment is relatively benign, it must be quantified to ensure vehicle integrity during nominal and off-nominal ascent conditions. Specialized Thermal Protection System (TPS) material is required to protect the vehicle from the extreme heating rates experienced during re-entry. The design of TPS requires convective and radiative heating environments for the entire vehicle surface, including localized heating rates on penetrations and protuberances.

We use a number of high fidelity codes to compute the flow field around the CM and LAS. Using multiple, independent codes for the same flight conditions increases our confidence in the computed results. The DPLR and LAURA codes are reacting Navier-Stokes solvers that include thermochemical nonequilibrium. They are used to compute aerothermodynamic heating rates and aerodynamic coefficients in the hypersonic regime. The NEQAIR radiation solver is a first principles physics code that computes the production of radiation by the gas in the hot shock layer, transport of the photons through the shock layer, and the radiative heating to the CEV surface. Aerodynamic coefficients in the subsonic, transonic, and supersonic regimes are computed using four different CFD tools. The OVERFLOW and USM3D codes solve the Reynolds-averaged Navier-Stokes equations using multiple overset structured grids. CART3D is an inviscid, compressible flow analysis package that uses Cartesian grids to solve flow problems over complex geometries such as the Orion Launch Abort System (LAS) with abort motor and attitude control motor plumes. The unstructured Euler CFD code FELISA is also being used.

Time-Accurate Computational Analysis of the Flame Trench Applications

Cetin C. Kiris*, Jeffrey A. Housman, Daniel Guy Schauerhamer**, Marshall Gusman**, William Chan*, and Dochan Kwak***

**NASA Ames Research Center, Moffett Field, CA 94035, USA*
(Tel: 650-604-4485; e-mail: Cetin.C.Kiris@nasa.gov)
***Eloret Inc Moffett Field, CA 94035, USA*

Abstract: Time accurate simulations are performed to analyze the effects of the exhaust plumes generated by the Space Shuttle's Solid Rocket Boosters (SRBs) on the Mobile Launch Platform (MLP) and flame trench. The subsequent ignition overpressure (IOP) waves are generated by the interaction of the plume with the trench. These IOP waves travel from the flame trench to the launch vehicle, and may cause stability problems during take-off. Computed results for one configuration of the Space Shuttle (STS-1) and three MLP configurations for a single SRB (used to represent Ares-Ix) are compared. Additional simulations are then performed to study pressure history at multiple points in the flame trench, in order to assist in wall repair efforts.

Keywords: Ignition Overpressure, Time-accurate calculations, Launch Environment.

1. INTRODUCTION

The purpose of this study is to characterize the ignition overpressure phenomenon during takeoff of new and existing launch vehicles. During ignition of the rocket propulsion system, transient pressure waves are initiated by the interaction of the exhaust plume with the flame trench. These ignition overpressure waves are generated during the initial buildup of thrust, in which mass is suddenly injected into the confined volume of the flame trench under the launch platform. The additional mass displaces the air within the trench causing a piston-like action and produces compression waves that travel up and down the trench. The traveling compression waves, along with their reflections, generate a series of strong pressure waves that travel back through the inlet of the trench towards the launch vehicle, where they may affect the stability of the vehicle during takeoff. For more details on the ignition overpressure phenomenon see (Jones, 1982). In order to assess the effects of the ignition overpressure waves on new and existing launch vehicles, time accurate simulations of the flame trench are performed for the Space Shuttle configuration and various MLP configurations with a single SRB, used to represent a preliminary design for Ares Ix. The SRB nozzle conditions are impulsively started with full thrust conditions (physically these conditions are achieved in approximately 0.3 seconds) and it is observed that IOP waves obtained from the Shuttle simulation correlate well with STS-1 flight data (the water suppression system was not implemented during the STS-1 launch), (R.S. Ryan, 1981). The impulsive start conditions are used for STS-1 and Ares-Ix simulations in order to assess the magnitude of the IOP waves. CFD simulations for the single SRB are performed for each configuration and a trend analysis of the IOP behavior is assessed.

Subsequent simulations improve the accuracy of IOP generation by including the time-accurate ramping effect of SRB ignition. For validation, pressure histories at points on the MLP and flame trench are compared to flight data from STS-4, with additional points that overlap the damaged wall regions reported to support wall repair efforts.

2. METHODS & APPROACH

2.1 Computational Model

The computational geometry for the launch site simulations includes the flame trench, the surrounding ground terrain, the Mobile Launch Platform, two plume deflectors, and the launch vehicle. Launch vehicles used in the simulations include a simplified Space Shuttle configuration with one external tank (ET) and two SRBs, and a preliminary Ares-Ix configuration consisting of one SRB rocket. The MLP for the Space Shuttle configuration contains two openings for the SRB plumes. For the single SRB configuration, various MLP options were investigated including either one of two openings, and with one or two deflectors. In order to provide high fidelity simulations of the plumes, various support structures in the MLP opening are modeled in the computational geometry and grid systems.

Structured viscous overset grid systems were built to model the different launch site configurations described above. A grid generation script based on the Chimera Grid Tools (CGT) script library, see (Chan, 2005), was developed to create the various grid systems. The overset grid and scripting approach are particularly well suited for this problem since they facilitate easy modifications to the grids to accommodate different options for the launch vehicle, MLP and deflectors. The Space Shuttle grid system contains 129 grids and 92 million grid points, see Fig. **1**. Grid systems for the single SRB with various MLP options contain 92 to 120 grids, and 73 to 87 million grid points.

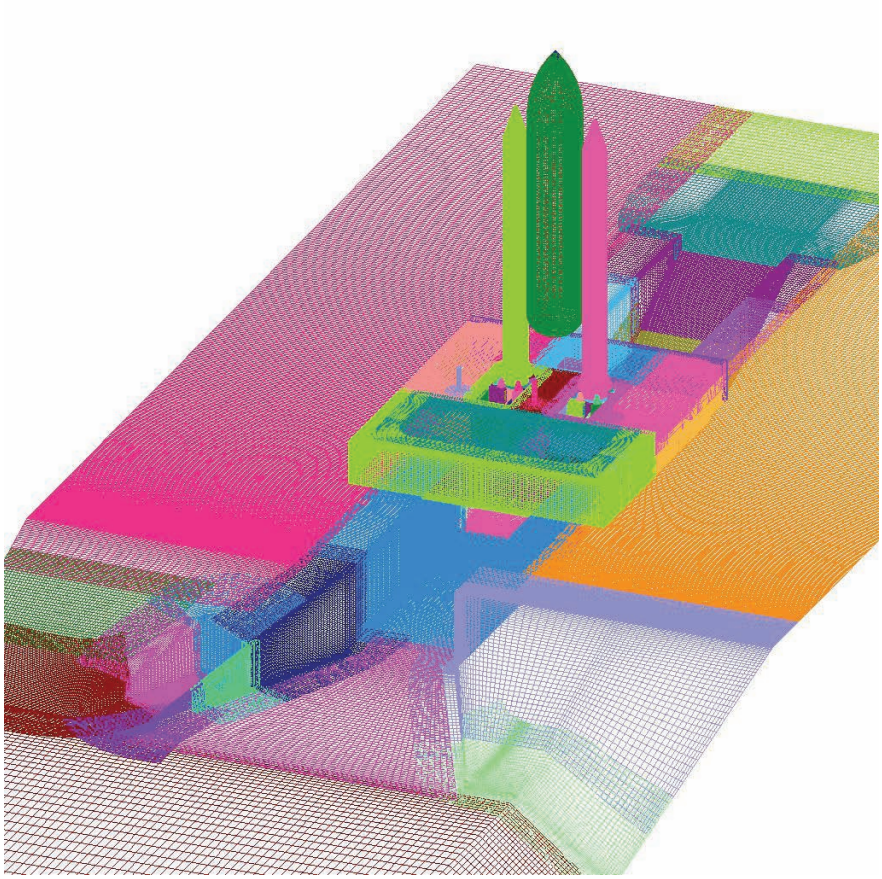


Fig. 1 Overset grid system of Shuttle external tank, SRBs, MLP, and flame trench.

2.2 OVERFLOW Solver

The CFD code OVERFLOW developed at NASA, see (P.G. Buning, 2003), is used in simulating the exhaust plume interaction with the flame trench. OVERFLOW is an implicit structured overset Reynolds Averaged Navier Stokes (RANS) solver. For the results reported here second-order central differencing with explicit artificial dissipation was used along with dual time stepping and the Spalart-Allmaras one equation turbulence model. Physical time steps on the order of 10^{-5} seconds and 20 to 40 subiterations per time step of the diagonalized implicit scheme were chosen to accurately represent the pressure waves and converge the numerical solution. The subiteration procedure consists of a right hand side evaluation followed by inversion of the diagonalized form of the approximate factored left hand side operator. The solver is made parallel through domain decomposition and uses the MPI message passing standard. The reported results were run on the Columbia supercomputer, at NASA Ames Research Center, on 128 processors. The overall simulation of two seconds of physical time required several weeks of wall clock time.

3. COMPUTED RESULTS

3.1 STS-1: Initial Validation

In order to validate the geometric model and computational procedure, the IOP waves generated during ignition of STS-1 (without the water suppression system) was simulated first. The physics of the IOP phenomenon for this configuration has been well analyzed and documented as in (R.S. Ryan, 1981). Instantaneous pressure contours of the IOP waves along with temperature contours of the exhaust jets are displayed in Fig. **2**. From these contour plots it is observed that large pressure waves are reflected from the trench, travel back towards the SRBs, and along the sides of the launch vehicle. Additionally, complicated vortical structures are observed as the plume enters the trench. In Fig. **3** the pressure at a point on the launch vehicle is plotted versus time, where the recorded flight data is on the left and the current CFD prediction is on the right. Good agreement between the peak pressure levels is observed, qualitative agreement is also good, provided the acoustic noise is removed from the flight data.

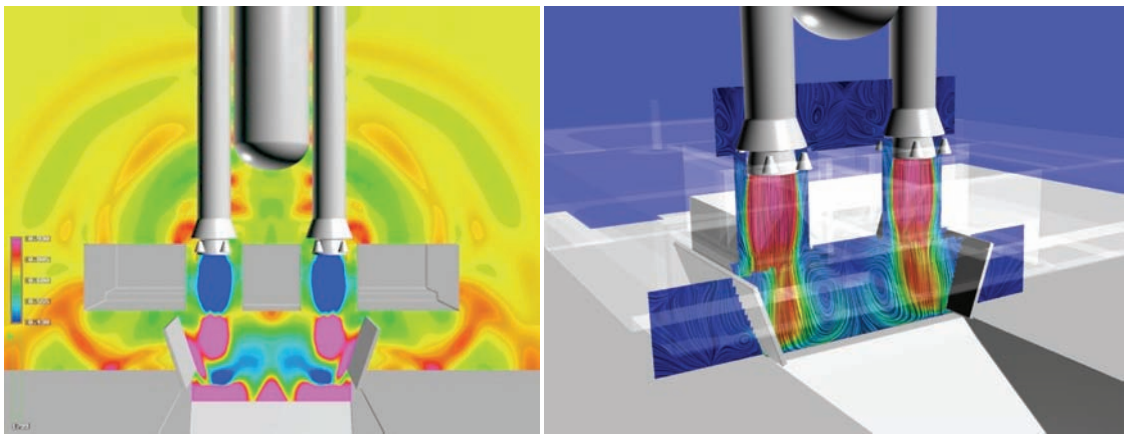


Fig. 2 Instantaneous IOP waves (left) and temperature contours (right) for the Shuttle configuration.

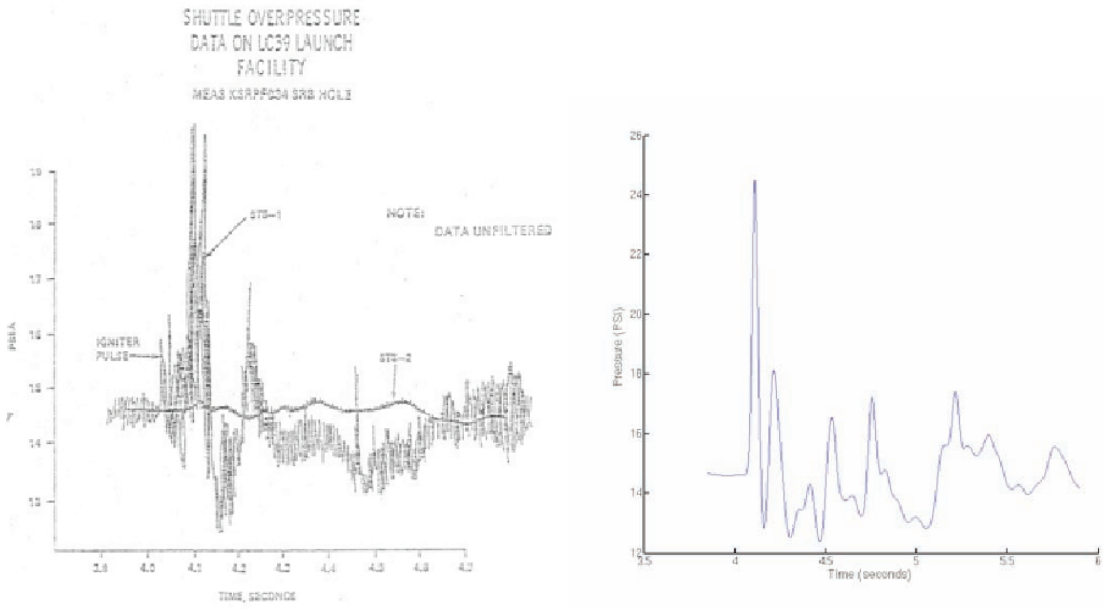


Fig. 3 STS-1 IOP comparisons between flight data, (R.S. Ryan, 1981) (left) and CFD prediction (right).

3.2 Ares 1-X: MLP Exhaust Hole Trade Study

In preparation for the launch of the Ares 1-X test-vehicle, a trade study was performed to investigate the feasibility of using a modified Shuttle MLP in place of the as-yet-unfinished Ares ML. The good correlation between the predicted CFD results and the recorded flight data for STS-1 gave confidence in the computational model. The model was modified to study the effects that different MLP configurations have on the IOP waves for the Ares 1-X (represented here by a single modified SRB, designated 'left SRB'). Deflectors are located below each hole to direct flow into the trench, and in all simulations, only the left SRB is present. The purpose of this study is to analyze the IOP phenomenon for three MLP configurations:

1. Both holes open, both deflectors present
2. Both holes open, right deflector removed
3. Right hole closed, right deflector removed

Fig. **4** shows comparison of predicted IOP waves on inner side of SRB for STS-1 and single SRB using three MLP configurations. It is observed that in each configuration the IOP waves with the peak values are reflected from the SRB's own exhaust hole. These waves sweep halfway up the SRB before the second weaker set of waves from the right side hole reaches it. This indicates that blocking the hole and removing the deflector do not have significant effects in reducing the IOP wave effects on the test-vehicle (as long as a proper water suppression system has been employed). The removal of the right side deflector allows the shock to penetrate sideways, thus delaying and slightly weakening the wave coming up the right hole. However, removing the right side deflector did not have significant reductions on the IOP pressure values on the vehicle. This also indicates primary contributions are coming from the left exhaust hole.

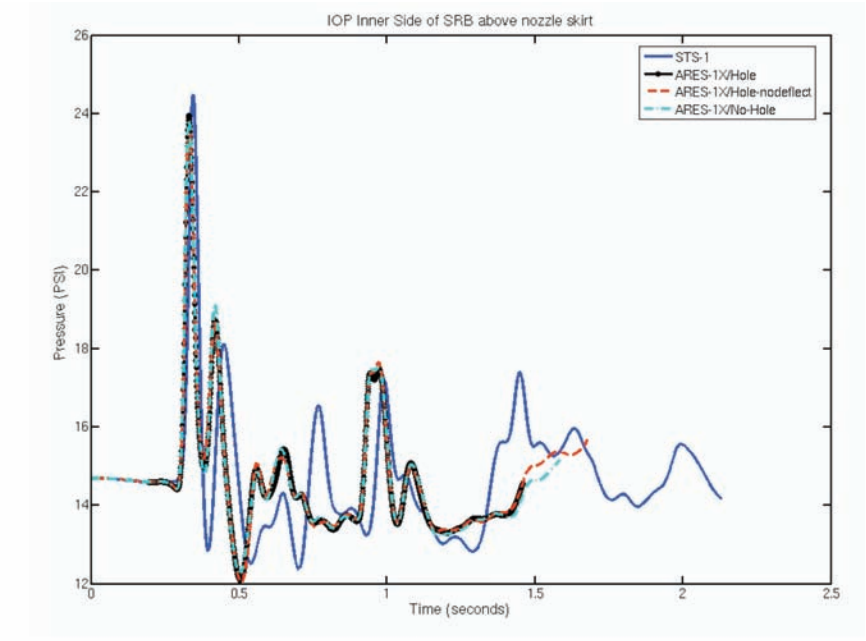


Fig. 4 Comparison of predicted IOP waves on inner side of SRB for STS-1 and single SRB using three MLP configurations.

4. SUMMARY

Time-accurate CFD simulations of the launch pad flame trench were presented using the Shuttle configuration and three different MLP configurations for a single SRB. The predicted IOP waves compared well with flight data for the Shuttle configuration. Computations for the single SRB showed similar IOP patterns for each MLP configuration and the STS-1 results. Wall damage investigations also compared well with flight data in the single-species model. Adding the water suppression system will likely reduce the IOP effects, but requires the addition of multiphase capabilities into the CFD model, and is the subject of future study.

REFERENCES

Jones, J. (1982). Scaling of Ignition Startup Pressure Transients in Rocket Systems as Applied to the Space Shuttle Overpressure Phenomenon. *APL JANNAF 13th Plume Technology Meeting*, (pp. 371-392).

R.S. Ryan, J.H. Jones, S.H. Guest, H.G. Struck, M.H. Rheinfurth and V.S. Verderaime (1981). *Propulsion Systems Ignition Overpressure for the Space Shuttle.* TM-82458, NASA.

Chan, W.M. (2005). Advances in Software Tools for Pre-processing and Post-processing of Overset Grid Computations. *Proceddings of the 9th International Conference on Numerical Grid Generation and Field Simulation.*

P.G. Buning, D. C. Jespersen, T.H. Pulliam, G.H. Klopfer, W.M. Chan, J.P. Slotnik, S.E. Krist and K.J. Renze (2003). *OVERFLOW User's Manual Version 1.8aa.* Internal Report, NASA.

PARALLEL CFD IN SHIP AERO AND HYDRODYNAMICS

A Parallel Hybrid Finite Element/Volume Methods for Ship Hydrodynamics

Shahrouz Aliabadi*, Tian Wan, Christopher Bigler*****

** Professor and Director, Jackson State University,*
Northrop Grumman Center for High Performance Computing, Jackson, MS, 39204
(Tel: 601-979-1821; e-mail: saliabadi@jsums.edu)
*** Postdoctoral Research Associate, Jackson State University,*
Northrop Grumman Center for High Performance Computing, Jackson, MS, 39204
(e-mail: tian.wan@jsums.edu)
**** Graduate student University of Michigan,*
Department of Naval Architecture and Marine Engineering, Ann Arbor, MI, 48109

Abstract: The following is a set of guidelines for preparing the final version of extended abstracts that have been accepted for presentation at the 2009 Parallel CFD conference. Use this document as a template to prepare the manuscript when using Microsoft Word 6.0 or later. Otherwise, use this document as an instruction set. For upload instructions, follow directions on the conference website.

Keywords: Include a list of 3-5 keywords, separated by commas.

1. INTRODUCTION

Recently we developed a hybrid finite element/volume (FE/FV) solver [1] for incompressible flows. The hybrid solver is based on the well-known pressure correction (projection) method [2, 3]. The solution procedure follows a segregated approach to decouple the pressure from the velocity. The velocity field is updated by solving the momentum equation provided that a known pressure field is given as a source term, through a cell- centered finite volume (FV) discretization. The pressure does not directly enter the momentum equation. Instead, an auxiliary variable, which is closely related to the pressure, takes the place of pressure in the momentum equation, providing pressure gradient information. We put the auxiliary variable on the vertices of cells. This deployment provides a convenient way to evaluate the pressure gradient using the local finite element basis functions. The incremental value of the auxiliary variable is computed by solving a Poisson equation using the Galerkin finite element (FE) method. The auxiliary variable is then used to update the velocity field. After the final velocity field is determined, the pressure can be updated using the auxiliary variable and the velocity divergence field. The pressure is updated in such a way that the pressure field is free of unphysical conditions in the boundary layer.

Our hybrid finite volume/element solver is aimed to take advantage of the merits of both the FV and the FE methods and avoid their shortcomings. For example, highly-stretched cells (also known as high-aspect-ratio cells) are commonly used inside the boundary layer for high Reynolds number flows to resolve the boundary layer and reduce the number of cells. The stabilization parameters in the stabilized FE based flow solvers [4, 5] are related to the characteristic element length that is not well defined for high-aspect-ratio mesh elements. Due to this, it is very difficult to control the numerical dissipation of stabilized finite element solvers. By contrast, the finite volume flow solver is very insensitive to the aspect ratio of the mesh cells. It is quite common for the FV solvers to handle cells with aspect ratios in the order of thousands [6, 7]. For this reason, we use the finite volume method to solve the momentum equation. On the other hand, the classic Galerkin FE method is very suitable for the elliptic typed equations like the pressure Poisson equation emerging from the segregated approach. Therefore, the combination of the FV method and the FE method is expected to perform well in the incompressible flow solvers based on the pressure projection method, which has been confirmed by our earlier work [1]. The numerical examples we presented in [1] are all about low Reynolds number flows. In our recent work, we have extended the hybrid flow solver to high Reynolds number flows using hybrid meshes with high aspect ratios [8].

We also incorporated the Detached Eddy Simulation (DES) [9] turbulence model into the flow solver to compute the eddy viscosity. The DES model was originally proposed to be an affordable hybrid Reynolds-averaged Navier-Stokes (RANS) and Large Eddy Simulation (LES) models for flows at realistic Reynolds numbers. In attached

boundary layers the DES model acts as a RANS model and in massively separated regions the DES functions as the LES model. A modified distance to the wall named the DES distance acts as a switch between the RANS mode and the LES mode [9]. There are two DES-based turbulence models. One is the one-equation Spalart-Allmaras (SA) DES model [9-10]. The other is the two-equation Shear Stress Transport (SST) DES model [11]. The SA-DES model is gaining more popularity due to its simplicity and fair accuracy. In our hybrid flow solver, we incorporated the SA-DES turbulence model. For details of implementations, see [8].

Generally, there are two distinct approaches in the numerical simulation of two-fluid flows (excluding panel methods]). Depending on the physical characteristics of the problem, either "moving-mesh" or "fixed-mesh" techniques are used. In the moving-mesh techniques, the motion of the free-surface is absorbed by moving the computational nodes located on the free-surface. Most of the moving-mesh techniques are based on either the space-time finite element formulations or the Arbitrary Lagrangian-Eulerian (ALE) formulations. In the applications where the deformation of the free-surface is large, the moving-mesh techniques usually result in element distortions. As the element distortions grow and become unacceptable, the generation of a new mesh and the projection of the solution from the old mesh to the new one is essential. In complex 3D applications, this procedure is extremely difficult and time consuming. In such cases, computations using fixed-mesh techniques are more desirable.

The most common fixed-mesh techniques are based on the VOF, the level-set and Interface-Sharpening/Global Mass Conservation (IS-GMC) methods. In these methods, the Navier-Stokes equations are solved over a non-moving mesh. A scalar function (or color function) acts as a marker to identify the location of the free-surface. This function is transported throughout the computational domain with a transient advection equation.

In this presentation we will describe the extension our implicit hybrid finite element/volume solver to ship aero-hydrodynamics. In our implementation, we will focus on the free surface sharpening strategy to minimize the smearing of the interface over time. For three-dimensional problems, iterative methods are almost mandatory in parallel implementation. The Generalized Minimal RESidual method (GMRES) has been widely used to solve large sparse systems. Because the GMRES algorithm involves only matrix-vector multiplication, it is unnecessary to form the Jacobian explicitly. The details of forming the Jacobian-vector production can be found in our previous paper [7].

The present solver has been parallelized on clusters using the ParMETIS mesh partitioning and the MPI parallel programming module. Due to the data structures we use, we have communications requirement for nodes, faces and cells. Very efficient non-blocking MPI functions are called to set up the inter-processor "gather" and "scatter" routines in the pre-processing stage. Scaling test has been taken on an eight-core cluster. The cluster has ten nodes each containing two quad-core Intel Xeon processors at 3.0 GHz. Each node has 32 GB memory, and the nodes are connected by Infini-band. Intel Fortran and C compilers are used, together with OpenMPI. OpenMPI is fully compatible with Infini-band, and as a result the communication latency of our code is very low. The scaling test is performed on a mesh of 25.2 million elements. Figure 1 shows the test results on 4, 8, 16, 32 and 64 processors. The CPU time of 4 processors is used as the baseline to compute speedup. The results show super-linear speedup. Figure 2 shows the CPU time breakdown on 4 and 64 processors, which shows that the time spent on MPI communication increases slightly from 0% to 2% when number of processors increases from 4 to 64. This explains why our code scales very well.

2. NUMERICAL EXAMPLES

Composite High-Speed Vessel. Northrop Grumman Ship Systems (NGSS) is currently researching the affects of adding a blended wing body system, which incorporates lifting bodies and hydrofoils, to a composite high-speed vessel (CHSV). In modifying the composite monohull to include the blended wing body system, NGSS has included sponsons and aft amas into the concept. The principle dimensions of the vessel are shown in Table 1.

The present code is used to perform CFD full-scale predictions and model test predictions of the CHSV at maximum design speed. The computational mesh has around 25.2M elements, and Figure 3 shows the mesh on the symmetry plane. In our final paper, a much refined mesh will be used and the computed results will be compared with those of the present mesh. Analysis was completed at a 3.677m static draft. Again, the speed of flow is 40 knots. The vessel displacement is 2000 metric tons, yielding a ship weight of approximately 1.96×10^7 Newtons. The vertical center of gravity is 4.68 meters above baseline and the longitudinal center of gravity is 27.95 meters forward of the transom. The temperature of the water is 15 degrees Celsius and the water density is 1025 kg/m^3. Figure 4 plots the pressure distribution on the vehicle. The lift/weight ratio is computed to be 1.19, which means the lift generated by this

model is 19% greater than the weight of the vehicle. Figure 5 plots the time history of the dimensionless lift force generated by both pressure and viscous force. It shows that the lift force has reached its steady state.

Table 1: CHSV Principle Dimensions

Length Overall	90.60 meters
Length Between Perpendiculars	73.58 meters
Maximum Beam	22.30 meters
Design Draft	3.677 meters
Design Displacement	2,000 metric tons
Maximum Design Speed	40 knots

ACKNOWLEDGEMENTS

The authors appreciate the support they received for this work from Northrop Grumman Ship Building.

REFERENCES

1. Tu S, Aliabadi S. Development of a hybrid finite volume/element solver for incompressible flows on unstructured meshes. *International Journal of Numerical Methods in Fluids* 2007; 55(2):177–203.
2. Guermond JL, Minev P, Shen J. An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering* 2006; 195:6011–6045.
3. Timmermans LJP, Minev PD, van de Vosse FN. An approximate projection scheme for incompressible flow using spectral elements. *International Journal of Numerical Methods in Fluids* 1996; 22:673–688.
4. Aliabadi S, Johnson AA, Abedi J. Stabilized-finite-element/interface-capturing technique for parallel computation of unsteady flows with interfaces. *Computer & Fluids* 2003; 32:535–545.
5. Aliabadi S, Tu S, Watts MD, Ji A, Johnson AA. Integrated high performance computational tools for simulations of transport and diffusion of contaminants in urban areas. *International Journal of Computational Fluid Dynamics* 2006; 20(3–4):253–267.
6. Delanaye M. Polynomial reconstruction finite volume schemes for the compressible Euler and Navier-Stokes equations on unstructured adaptive grids. Ph.D. thesis, Universite de Liege, 1996.
7. Tu S, Watts MD, Fuller A, Patel R, Aliabadi S. Development and performance of CaMEL_Aero, a truly matrix-free, parallel and vectorized unstructured finite volume solver for compressible flows. *Proceedings of the 25th Army Science Conference*, Orlando, FL, 2006.
8. Tu S, Aliabadi S, Patel R, Watts M. An implementation of the Spalart-Allmaras DES model in an implicit unstructured hybrid finite volume/element solver for incompressible turbulent flow. To appear in the International Journal of Numerical Methods in Fluids.
9. Spalart PR, Jou W-H, Strelets M, Allmaras SR. Comments on the feasibility of LES for wings and on hybrid RANS/LES approach. *Proceedings of the 1st AFOSR International Conference on DNS/LES*, Columbus, OH, 1997.
10. Nikitin NV, Nicoud F, Wasistho B, Squires KD, Spalart PR. An approach to wall modeling in large-eddy simulations. *Physics of Fluids* 2000; 12(7):1629–1632.
11. Strelets M. Detached eddy simulation of massively separated flows. AIAA paper 2001–0879, the 39th AIAA Aerospace Sciences Meeting and Exhibits, Reno, NV, 2001.

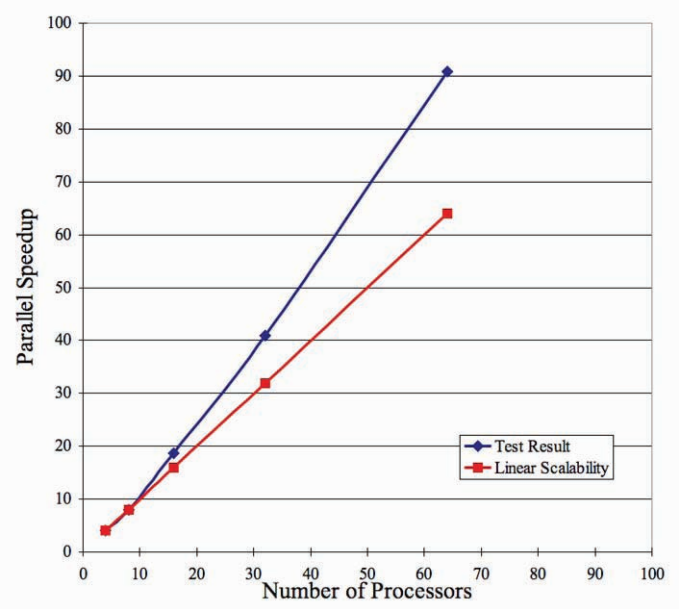


Fig. 1: Parallel scalability test results..

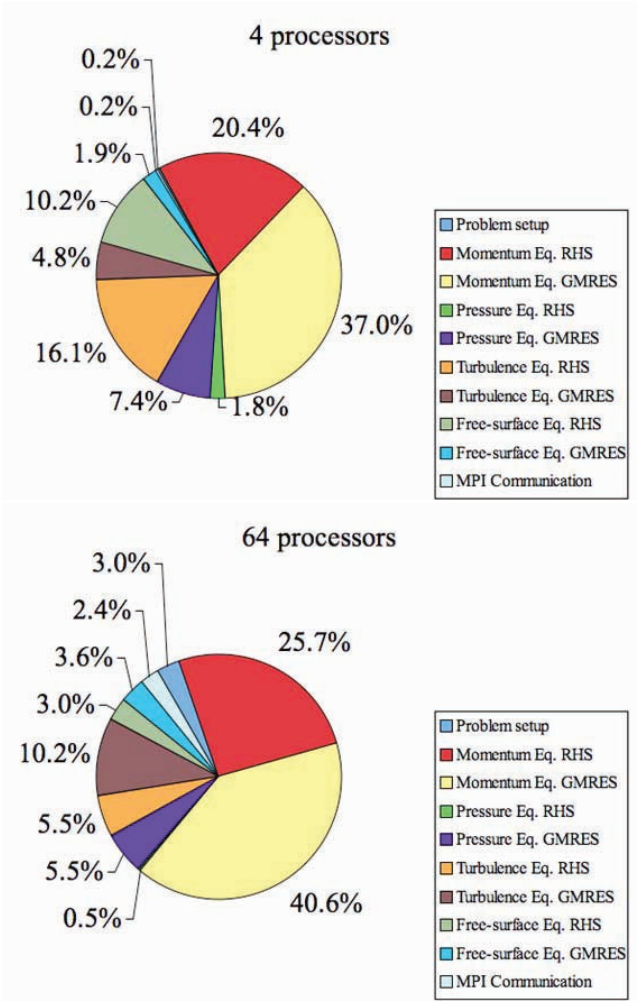


Fig. 2: Pie-chart of the scaling test results. The values plotted are the CPU time percentage.

(Top) 4 processors. (Bottom) 64 processors.

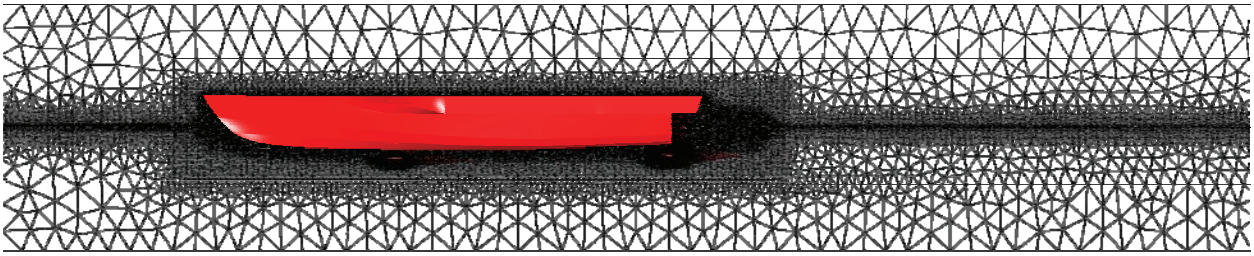


Fig. 3: Computational mesh on the symmetry plane. Note that the mesh is clustered near the vehicle and the free surface. The size of the mesh is around 25.2M elements.

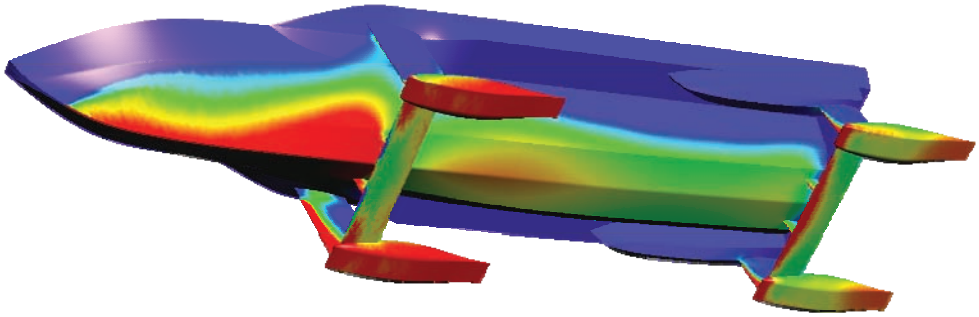


Fig. 4: Pressure contours on the vehicle body.

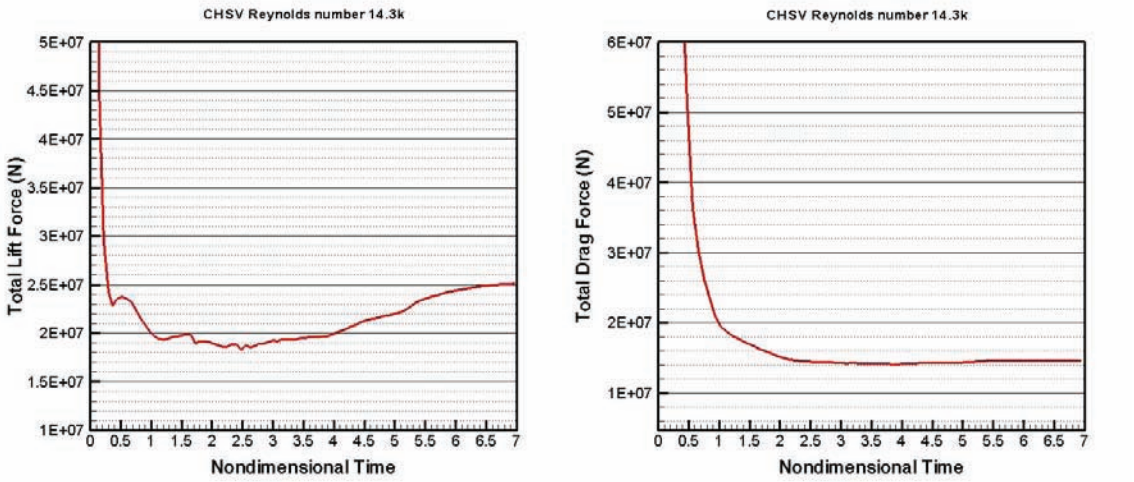


Fig. 5: Time history of total lift and drag force.

On the Parallelization of Particle Finite Element Method

Pooyan Dadvand*, Riccardo Rossi, Eugenio Oñate *****

** CIMNE, C1 Building, North Campus-UPC, Gran Capitan, s/n, 08034, Barcelona, Spain*
(Tel: +34 93 401 60 38; e-mail: pooyan@cimne.upc.edu)
*** CIMNE, C1 Building, North Campus-UPC, Gran Capitan, s/n, 08034, Barcelona, Spain*
(Tel: +34 93 401 60 38; e-mail: rrossi@cimne.upc.edu)
**** CIMNE, C1 Building, North Campus-UPC, Gran Capitan, s/n, 08034, Barcelona, Spain*
(Tel: +34 93 401 60 35; e-mail: onate@cimne.upc.edu)

Abstract: The solution of problems involving free surface effects and large deformations of the computational domain constitutes a challenging problem. As examples we may consider the analysis of coastal structures subjected to the action of waves or the behavior of ships in heavy sea conditions. The Particle Finite Element Method (PFEM) provides an interesting approach for the solution of such problems.

The key feature of PFEM is the use of a lagrangian approach for the computation of the fluid behavior together with a alpha-shape technique for the detection of the free surface. The use of a lagrangian approach for the solution of the fluid makes the remeshing process a necessary part of solution.

Several existing approaches for the parallel solution of CFD problem are also applicable to PFEM [4, 2] fluid solver. On the other hand the remeshing process and the dynamic changes of the domain make its parallelization more complex than for traditional fixed mesh approaches. Since this method consists of several remeshing step a robust and fast parallel remeshing algorithm is required. A dynamic load balancing is an important complementary to the system in order to deal with moving meshes.

In this presentation first we give a brief description of PFEM and its main advantages. Then we continue describing the main difficulties and their solutions in parallelization of PFEM for shared memory and distributed memory machines.

Keywords: Particle Finite Element Method, Parallelization, Remeshing.

1. INTRODUCTION

The simulation of problems involving large free surface effects or significant variation in the shape of the computational domain, can be performed via a number of different techniques. Eulerian approaches such as level-set or volume-of-fluid, provide a computationally effective tool for the solution of a large variety of problems in the field. There exists however a number of problems for which alternative techniques need to be looked for. It has been proven once and again that Lagrangian techniques are able to fill this gap, thanks to their ability to track accurately the motion of particles and to apply exactly the B.C. on the free surface. The Particle Finite Element, which is the tool used in current work, is an example of such approaches.

The basic idea relies on coupling the mathematical tools provided by the FEM with an aggressive Delaunay remeshing strategy that allows dealing with the changes in the computational domain. The characteristics of the method, and in particular the usage of Implicit solvers for the description of the fluid flow, make it very well suited for the simulation of problems involving large changes in the physical properties. The combination of FE solvers with remeshing tools on the other hand allow a fully modular approach, which allows taking advantage of the implementation effort made in the different specialized solvers. Unfortunately the nature of the formulation, which is based on a dynamically changing data structure, makes its parallelization demanding due to the needs of dynamically adapting the connectivity and the matrices used in the solution.

The specific purpose of this work is to describe the progress in the parallelization of the technique, describing in particular the strategy that is currently being used towards an incremental parallelization of the method.

2. PFEM

The Particle Finite Element Method is essentially based on the steps:

- Lagrangian Prediction of the node Position
- Meshing step and improvement of the existing mesh
- Arbitrary Lagrangian Eulerian Solution of the problem on the top of the newly created mesh

Both step 1 and step 3, require a solution of a differential problem, which is performed using a standard implicit finite element technology. Steps 2 requires a Delaunay reconnection of an existing cloud of nodes and the pass of a "Alpha-Shape" geometrical filter for the identification of the boundaries. The computational effort needed for the solution of the first and last steps can be estimated to be $O(N^{\alpha})$ with $\alpha=1.3\sim1.5$ in our problems while the one for the Delaunay reconnection can be proved to be $O(N\ logN)$. The immediate implication is that the time spent in the solution steps will dominate the overall solution cost for large problems. This ``theoretical'' prediction is proved by the measurements shown in Fig 1. This simple observation allows one to conclude that some significant advantages in the solution of large scale problems can be obtained by parallelizing the single field solvers involved. This lead to the choice of a two steps parallelization effort, in which the implicit fluid solver is parallelized first and the parallel remeshing is considered as a second step.

The first step is performed using the Trilinos Library, leading to the definition of a parallel solver on a dynamically varying mesh. The second step is currently being performed and it is described in the next section.

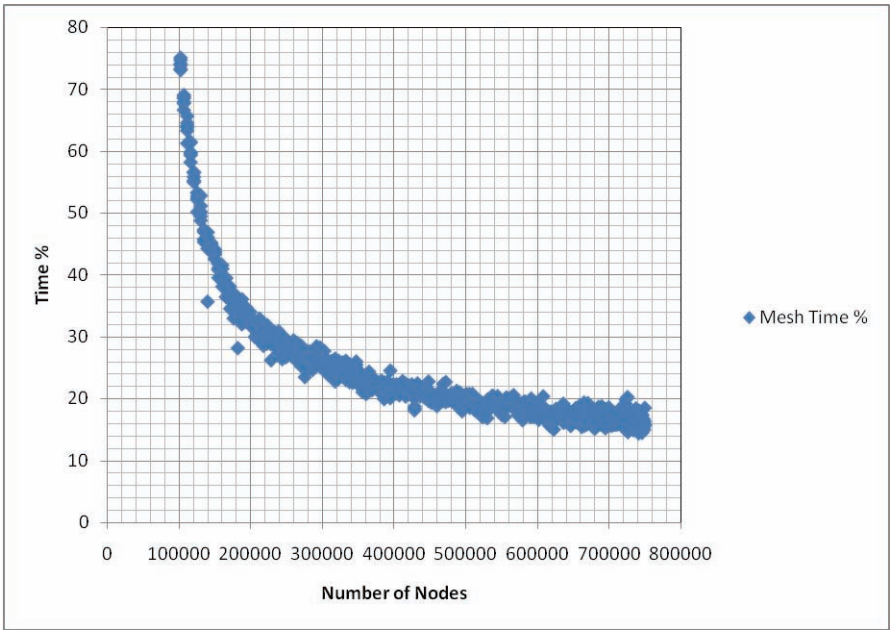


Fig. 1: *Remeshing time for different number of nodes.*

3. PARALLEL REMESHING

As we mentioned in previous chapter the remeshing process is an essential part of the PFEM. The percentage of the solution time spent by the remeshing process is highly depended to the number of the nodes in the model. Figure 1 shows the time percentage of remeshing process in relation with number of the nodes in the model. It can be seen that with increment of model size the relative time spend in remeshing decreases. Base on this fact a simple approach was to use a parallel solver with serial mesh generator. This approach can be used for small number of cpus but it is bounded to the limits of the Amhdal's law.

Even if the parallelization of Delaunay procedures is known to be an extremely difficult task, this is hopefully not the case for the reconnection of a cloud of existing nodes. A number of efforts in the field have been published in the recent past [3] at CIMNE and in other institutions. The technique that is currently being explored is based on a scalar oct-tree balancing of the cloud of nodes followed by a parallel remeshing of each of the domain separately. An overlap area is left unmeshed, and is covered by a scalar Constrained Delaunay step.

Another approach is using local optimization techniques like edge flipping [1] for reducing and even eliminating the global remeshing process. The localness of these operations can be exploited to ensure the scalability of these codes. The methodology consists of remeshing the interior part of each partition in parallel and the interfaces in scalar.

REFERENCES

1. George, P. L. and Borouchaki, H. (2003). Back to edge flips in 3 dimensions, *12th International Meshing Roundtable*, Sandia National Laboratories, 393-402.
2. Idelson, S.R. and Oñate, E. and Del Pin, F. (2004). The particle finite element method: a powerful tool to solve incompressible flows with free-surfaces and breaking waves. *International Journal for Numerical Methods in Engineering,* 61, 964 - 989.
3. Kohout, J. and Kolingerová, I. and Zára, J. (2005). Parallel Delaunay triangulation in E^2 and E^3 for computers with shared memory. *Parallel Computing,* 31 (5), 491 - 522.
4. Oñate, E. And Idelson, S. and Del Pin, F. and Aubry, R. (2004) The Particle Finite Element Method. An Overview, *International Journal of Computational Methods*, 1(2), 267-307

Large Scale Parallel Computing and Scalability Study for Surface Combatant Static Maneuver and Straight Ahead Conditions Using CFDShip-Iowa

Frederick Stern, Shanti Bhushan, Pablo Carrica and Jianming Yang

IIHR-Hydroscience and Engineering
The University of Iowa
Iowa City, IA 52242-1585

Abstract: Scalability studies are performed for CFDShip-Iowa URANS/DES curvilinear (V4) and Cartesian (V6) solvers and calculations are performed using largest grids to date. V4-DES solution for DTMB 5415 with bilge keels at 20° static drift on 250M grid is analyzed including the instantaneous and mean separation flow patterns, vortical structures and associated instabilities, and turbulent kinetic energy (TKE) distribution and budget. V6 RANS solution of 5415 bare hull at straight ahead condition on 276M grid is compared with benchmark V4-blended *k-ω/k-ε* **(**BKW), V4-wall-function (WF), V4-DES results and EFD data to identify limitations of wall-layer modeling using WF for immersed boundary method (IBM).

1. INTRODUCTION

Large scale computations are important for ship flows as they enable resolution of small-scale physics, improve our understanding of turbulent and vortical structures, two phase flows and air entrainment, help in identifying modeling issues and develop better models. Herein, calculations are performed using largest grids to date for CFDShip-Iowa V4 and V6 solvers. The objective of this paper is to study scalability and identify scalability bottlenecks, and to validate V4 and V6 predictions. The scalabilities of V4 and V6 are studied up to 2300 processors. V4 simulation is performed for surface combatant 5415 with bilge keels at 20° static drift on 250M grid. The study focuses on analysis of the turbulent and vortical structures, associated instabilities including verification and validation (V&V). V6 simulation is performed for 5415 bare hull at straight ahead condition on a 276M grid using WF with $y^+ = 30$ to identify limitations of wall-layer modeling using WF for IBM. For this case, V4-BKW, -WF simulations on 615K grid and -DES on 300M grid are also performed to obtain benchmark results.

The general-purpose code V4 solves the URANS/DES equations in the liquid phase of a free-surface flow using level-set method [1] in either absolute inertial earth-fixed or relative inertial coordinates. The turbulence modeling is performed using two-equation BKW or anisotropic Reynolds stress (ARS) models and has DES and WF options. A multi-block dynamic overset grid interpolation using SUGGAR is used to allow relative motions between the grids for six degrees of freedom (6DoF) ship motions. The governing equations are discretized using cell-centered finite difference schemes on body-fitted curvilinear grids and solved using a predictor-corrector method. The pressure Poisson equation is solved using the PETSc toolkit. Message Passing Interface (MPI) based domain decomposition is used for high performance computing (HPC). V4 has several simulation based design (SBD) functionalities for ship resistance, propulsion, seakeeping and maneuvering.

In V6 the URANS/DES/LES two-phase flow governing equations are solved in absolute inertial earth-fixed coordinates using IBM [2]. The interface modeling is performed using level-set, particle level-set or coupled level-set and volume-of-fluid (CLSVOF) method [3]. The turbulence modeling is performed using either BKW/blended *k-g/k-ε* (BKG) models with WF for URANS or a dynamic Smagorinsky model for LES. The governing equations are discretized using finite differences on a non-uniform staggered Cartesian grid and solved using a four-step fractional-step method. The pressure Poisson equation is solved using multi-grid PETSc or HYPER library. The code uses MPI based domain decomposition for solution on parallel processors and MPI-I/O. V6 is a research code which is being turned into a ship hydrodynamics code. The code has been previously applied to several fundamental problems such as wave breaking, bubble entrainment and air layer drag reduction. V6 has advantages over V4 for both accuracy and HPC. V6 requires about 50% less memory and 20-25% less CPU time/time-step/processor/grid-point than V4 for a similar size grid. In the long term V6 has the potential to develop into an efficient and robust general-purpose solver.

2. CFDSHIP-IOWA HPC OPTIMIZATION AND SCALABILITY STUDY

Carrica et al. [4] improved V4 HPC performance by optimizing serial and parallel execution of routines and overset grid assembly. Several inefficient routines were rewritten and unnecessary MPI barriers, collective MPI instructions were eliminated. Memory usage was improved to avoid shared memory and allow scalability on distributed memory machines. SUGGAR was implemented as a library such that it can be executed on a dedicated MPI rank.

V4 strong scalability tests are performed for fixed sinkage and trim 5415, Re=4.85×10^6, Fr=0.28 using a 115M grid on NAVO's IBM P6 DaVinci and CRAY XT5 Einstein. Weak scalability test were performed by Carrica et al. [4] on AFRL SGI Altix 4700 Hawk for fixed sinkage and trim KVLCC and 5415 geometries using 120K and 240K grid points per processors, respectively. Fixed sinkage and trim cases are chosen to avoid SUGGAR interpolation every time step. The solver CPU times are based on average of 10 time steps without I/O. As shown in Fig 1(a), V4 scales almost linearly up to 1024 processors and the speedup drops 37% below ideal scaling for 2048 processor on Einstein. Similar scaling is observed on DaVinci. On Hawk reasonable scalability is obtained only up to 512 processors. About 71% of the CPU time is spent for solving the pressure Poisson equation. A fairly good weak scalability is observed up to 500 processors as shown in Fig. 1(b), even for different problems and for grid points per processor ratio of 2:1. The deviation from ideal scaling for the forward speed diffraction case (Fr = 0.28) using 115M grid points is attributed to the saturation of memory on the nodes.

Yang et al. [5] extended the one-dimensional slab decomposition in V6 to three directions for parallelization and the inter-processor communications for ghost cells were changed into non-blocking mode. Parallel I/O using MPI2 was implemented such that all processors read from and write to one single file instantaneously.

V6 strong scalability tests are performed for fixed sinkage and trim 5415, Re=4.85×10^6, Fr=0.28 using 100M, 270M and 540M grids on DaVinci. Weak scalability tests are performed using 262K and 525K grid points per processor. The solver CPU times are based on average of 10 time steps without I/O. As shown in Fig. 1(a), V6 scalability drops by 25% and 33% below ideal scaling on 2048 processors for 540M grid and 270M grid on DaVinci, respectively. As expected, the scalability of the code improves when the grid size increases. The scalability on DaVinci is much better than that on Babbage, where performance drops by 20% even on 256 processors. Almost 92% of the CPU time is spent for the pressure solver, which dictates scalability of the solver. A weak scaling factor of 1 is expected as each processor handles the same amount of data. However, Fig. 1(b) shows that the CPU time increases with the number of processors. The deviation from ideal scaling is mainly affected by HYPRE, which requires extra levels of coarsening for finer grids leading to extra computational cost.

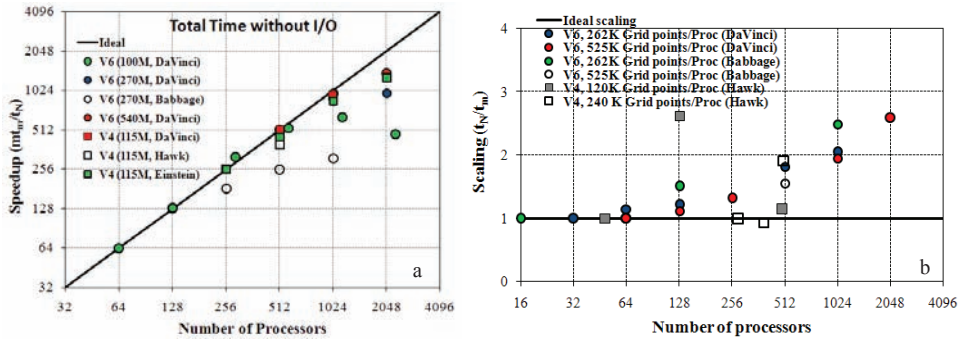


Figure 1: *(a) Strong scalability for V4 and V6 are compared with ideal scaling. (b) Weak scalability tests using 120K and 240K grid points per processor for V4 and 262K and 525K grid points per processor for V6.*

3. 5415 WITH BILGE KEELS AT 20° STATIC DRIFT CONDITION USING V4

SIMMAN workshop [6] included validation of 5415 at 10° static drift for which comparison error E=D-S, where D is EFD and S is solution, were 15.2%D, 3.0%D and 8.2%D for longitudinal force X_t, sway force Y_t and yaw moment N_t coefficients, respectively. Sakamoto [7] performed calculations for 5415 at 10° and 20° static drift and obtained E up to 12%D, 11%D and 2%D for X_t, Y_t and N_t, respectively. Sensitivity studies were performed using ARS, BKW and DES turbulence models and using fine grids up to 20M, which showed minimal improvements. One possibility for error was thought to be due to the absence of transition modeling in turbulence equations. Based on the URANS simulation here we construct a grid which has sufficient density to activate DES properly. E reduces significantly on this grid presumably due to correct prediction of cross-flow separation.

URANS/DES simulations are performed using BKW for fixed sinkage=1.92×10^{-3} and trim=0.136° 5415 with bilge keels at 20° static drift condition Re=4.85×10^{6}, Fr=0.28 and results are compared with EFD data [8]. The V&V study is performed following the quantitative methodology and procedures proposed by Stern et al. [9] using three systematically refined grids with $r_G = \sqrt{3}$ consisting of 10M, 48M and 250 M points.

Table 1 summarizes the V&V study for force and moment coefficients. The subscripts f, p and t correspond to frictional, pressure and total, respectively. $U_I \leq 0.75\%S_I$ for the force and moment coefficients on all the grids. For frictional coefficients $U_I/\varepsilon_{12}<0.12$, whereas for pressure coefficients $U_I/\varepsilon_{12}>2.7$. Monotonic convergence $R_G<0.52$ is obtained for X_f, X_p and X_t, oscillatory convergence $-0.62<R_G<0.12$ for Y_f, Y_p and Y_t, oscillatory divergence for N_f, and monotonic divergence for N_p and N_t. The correction factor $CF<0.65$ for X_f, and X_t, whereas CF=16.5 for X_p which is considered to be divergent. The observed order of accuracy p_G for X_t is 3.29 close to theoretical order of accuracy p_{Gth} of 4. U_G is 3.25%S_I for X_f and <1%S_I for X_t and sway resistance coefficients. Large U_I/ε_{12} values for pressure coefficients indicate contamination of U_G by U_I. U_G estimates could also be affected due the correlation of modeling and numerical errors in DES.

Table 1: *Resistance and moment coefficients verification study for 5415 with bilge keels at 20° static drift condition.*

Parameters	EFD	10M URANS	DES Verification Study			ε_{23}%	ε_{12}%	U_I%	U_I/ε_{12}	R_G	p_G	CF	U_G%	U_D%	U_V%
			Grid 3:10M	Grid 2: 48M	Grid 1: 250M										
$X_t\times10^2$	-2.865	-2.962	-2.879	-2.979	-2.996	3.33	0.55	0.38	0.691	0.164	3.29	0.64	0.44	7.9	7.92
	$\%E_X$	+3.39	+0.49	+3.98	+4.57	-									
$Y_t\times10^2$	15.29	16.631	16.088	15.821	15.860	1.68	0.22	0.74	3.364	-0.128	-	-	0.63	4.1	4.21
	$\%E_Y$	+8.77	+5.22	+3.47	+3.72	-									
$N_t\times10^2$	5.935	6.098	6.042	5.997	5.855	0.77	2.42	0.70	0.289	3.13	-	-	-	1.3	
	$\%E_N$	+2.75	+1.82	+1.05	-1.35	-									

$\varepsilon_{23}=|(S_2-S_3)/S_1|\times100$; $\varepsilon_{12}=|(S_1-S_2)/S_1|\times100$

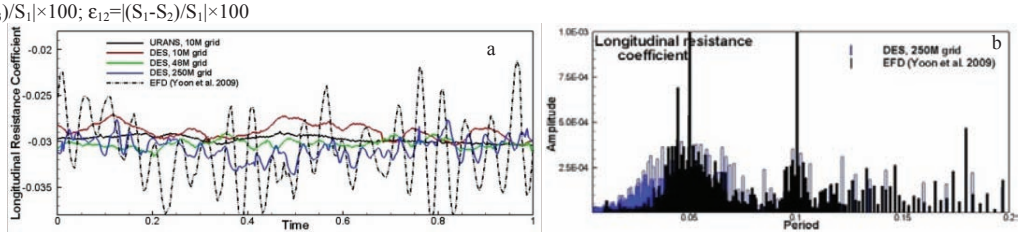
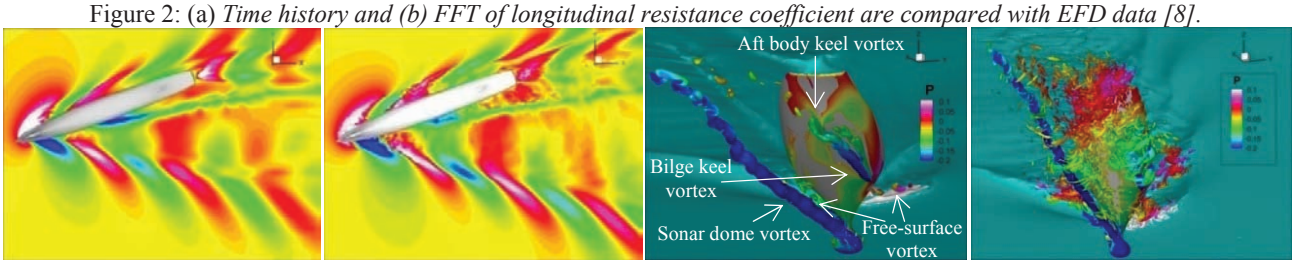


Figure 2: (a) *Time history and (b) FFT of longitudinal resistance coefficient are compared with EFD data [8].*



(a) 10 M grid DES (b) 250M grid DES (c) 10M grid DES (d) 250M grid DES

Figure 3: *(a), (b) Free-surface wave elevation and (c), (d) isosurfaces of Q=1000 obtained for static 20° drift 5415 DES simulations using 10M grid and 250M grids. Wave elevation contour levels are from -0.01 to 0.01 with intervals of 4×10^{-4}.*

DES shows 4-5% lower E than URANS, and the best results are obtained on 48M grid where E<4%D. Results show $U_G << U_D$, which suggests errors are dominated by modeling errors. The validation uncertainty U_V=7.92%D for X_t and 4.21%D for Y_t. E for the fine grid is 4.57%$D<U_V$ for X_t, and 3.72%$D<U_V$ for Y_t. Thus both X_t and Y_t are validated. $E\leq1.82\%D$ for N_t on all the grids, which is slightly larger than U_D=1.3%D. FFT of X_t on 250M grid shows two dominant frequency zones τ=0.05 and 0.1 as shown in Fig. 2. Peak amplitudes at these periods are also observed in the EFD data. However, experimentalists [8] attributed these frequencies to the noise from carriage speed vibrations τ=0.05 and load-cell τ=0.1.

Overall, the wave patterns are similar on all the grids. The small scale structure close to the hull and wave breaking patterns are resolved better on finer grids as shown in Fig. 3. Isosurfaces of Q in Fig. 3 for 10M grid shows sonar dome, bilge keel, aft-body keel, and port and starboard free-surface vortices. The sonar dome vortex exhibits helical instability breakdown. Kelvin-Helmholtz, Karman and flapping-type instabilities are observed on the leeward side.

Further analysis is required to obtain frequency and scaling of these instabilities. In the 250M grid simulation sonar dome vortex is clearly visible, but the other vortices are masked by the massive cross-flow separation.

4. 5415 BARE HULL AT STRAIGHT AHEAD CONDITION USING V6

Simulations are performed for 5415 bare hull at straight ahead condition Re=4.85×10^6, Fr=0.28 using V6-WF with BKG for the half ship at fixed sinkage=1.92×10^{-3} and trim=0.136° using 34M with y^+=150, 92M with y^+=60 and 276M with y^+=30 grids. V6-WF results are compared with half ship V4-BKW, V4-WF results on 615K grid, V4-DES on 300M grid and EFD [10] in Fig. 4.

V4-BKW over predicts resistance coefficient by 5.6%D, V4-WF predictions are within 2%D and V4-DES results are within 2.6%D. The poor V4-BKW predictions could be due to the coarse grid resolution as Sakamoto [8] has shown errors less than 3% on a 1.27M grid. Further study is required using finer grids to obtain V4-BKW and V4-WF benchmark solutions. The wave elevation patterns are predicted well in both V4 and V6 simulations on all the grids when compared with the EFD data. The wave pattern resolution improves with grid refinement. V4-DES and V6-WF with y^+ = 30 results are in detailed agreement with EFD.

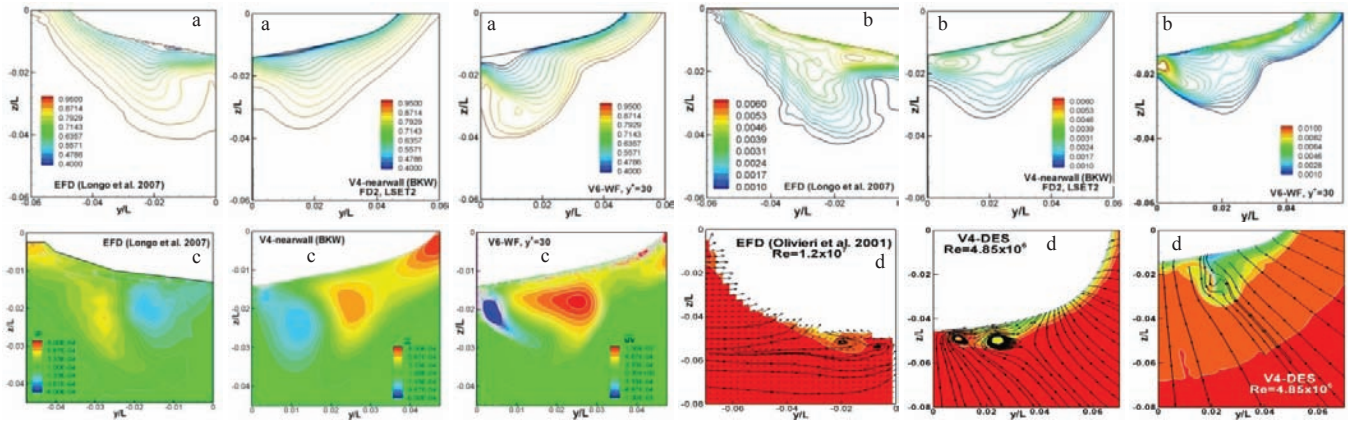


Figure 4: *Comparison of (a)streamwise velocity, (b) turbulent kinetic energy and (c) shear stress $\overline{u'v'}$ distributions on x/L=0.935 obtained using V4-BKW and V6-WF and (d) V4-DES boundary layer and wake profile at x/L=0.6 and 0.935with EFD data [10].*

EFD mean flow pattern shows interactions between the hull boundary layer and sonar dome (near center plane) and after body shoulder (near mid girth) out-board rotating axial vortices. For the nominal wake plane, inboard of the axial vortex center and near the center plane, high momentum fluid is transported towards the hull thinning the boundary layer, whereas outboard of the vortex center low momentum fluid is transported away from the hull causing bulge in the boundary layer. The mean and turbulent nominal wake flow pattern shows similarity to the boundary layer and turbulence structures in the presence of common-down streamwise vortex pair. V4-BKW predicts the sonar dome rotating vortex and its interaction with boundary layer fairly well. However, the sonar dome vortex is over predicted at development and the after body shoulder vortex is not resolved well suggesting rapid dissipation of the vortex. At the nominal wake plane the boundary layer is thicker at center plane compared to the EFD data and the bulge is underpredicted due to weak vortex strength. The TKE and stress contours compare well with EFD qualitatively but not quantitatively. V4-WF predictions are similar to that of V4-BKW for mean velocity profiles, but has penalty in prediction of turbulence quantities. V6-WF results improve with decrease in y^+ and has penalty similar to V4-WF. V6-WF with y^+=30 over predicts the bilge vortex strength significantly resulting in over prediction of boundary layer bulge. TKE peak is observed to be at the center plane compared to y/L=0.01 in EFD and the peak value is overpredicted by about 40%. Both normal and shear stresses are also over predicted. Preliminary analysis of the V4-DES instantaneous solution shows that it captures the co-rotating vortices at mid-girth and the bulge of the streamwise velocity is predicted better than V4-BKW at nominal wake plane.

5. CONCLUSIONS

Scalability study shows 37% and 25-30% speedup drop below ideal scaling on 2048 processors for V4 and V6, respectively. Weak scalability shows that the best results using V4 and V6 are obtained using 240K and 525K grid

points per processors, respectively. Thus it can be estimated that the largest simulation using V4 and V6 can be performed using 490M and 1billion grid points, respectively. The main scalability bottle neck is identified to be the pressure Poisson solver. Future simulations will be performed in close collaboration with software experts at ANL to address the scalability limitations of PETSc and HYPRE. Development of a domain decomposition version of SUGGAR is required to remove scalability limitations of V4 for dynamic motions.

V4-DES V&V study for 5415 with bilge keels at 20° static drift simulation shows grid convergence with $U_G < 3.5\%S_I$ for both X_t and Y_t, which are validated within $5\%D$ uncertainty. FFT of force and moment coefficients show dominant frequency at τ=0.05 and 0.1. These dominant frequencies are also observed in the EFD, but were attributed to experimental noise. The 250M grid results show massive cross-flow separation and highly unsteady wave pattern with bow-wave breaking. Analysis of the volume solutions will be performed to study mean flow to identify the vortical structures, TKE and stress distribution and TKE budget to understand the mean turbulent structures. Mean flow will be subtracted from the instantaneous flow to study the vortex instabilities and organized turbulent vortical structures. Future work for V&V includes study using intermediate grids, reduction of U_I and estimation of modeling and numerical error correlation in DES. The CFD results will guide the planned experiments for PIV flow field measurements. Load-cell measurements will be revisited to reduce experimental noise.

In the 5415 at straight ahead simulation, both V4-DES and V6-WF provide detailed agreement of the wave pattern. V4-BKW provides reasonable results for the mean velocity, TKE and stress profiles, but fails to predict the sonar dome vortex strength accurately. WFs show penalty in flow prediction both in V4 and V6, where the latter shows the worst comparison with EFD. Best results using WFs are expected with y^+=30, thus it can be concluded that wall-layer modeling using WF for IBM has limitations for complex geometries. Preliminary V4-DES results show significant improvements over the V4-BKW. Mean flow will be obtained from V4-DES to identify the mean vortical structures, and the turbulence quantities at nominal wake plane will be compared with EFD data. Future work includes validation of force computations using V6 and obtaining V4-BKW and -WF benchmark solutions on appropriate grid. Future development of V6 includes weakly and strongly coupled wall layer (curvilinear orthogonal and non-orthogonal grids) with Cartesian background grid.

ACKNOWLEDGEMENTS

The research is sponsored by the Office of Naval Research under Grant N00014-01-1-0073 and N00014-06-1-0420, administered by Dr. Patrick Purtell. The computations were performed at DoD NAVO HPC machines.

REFERENCES

1. Carrica, P.M., Wilson, R.V., Noack, R. and Stern, F. (2007). Ship Motions Using Single-Level Set with Dynamic Overset Grids. *Computers and Fluids*, **36**(9), 1415-1433.
2. Yang, J. and Stern, F. (2008). Sharp interface immersed-boundary/level-set methods for wave-body interactions. Submitted to *Journal of Computational Physics*.
3. Wang, Z., Yang, J., Koo, B., and Stern, F. (2009). A coupled level set and volume-of-fluid method for sharp interface simulation of plunging breaking waves. *International Journal of Multiphase Flow*, **35** (3), 227-246.
4. Carrica, P.M. et al. (2008). Large-scale DES computations of the forward speed diffraction and pitch and heave problems for a surface combatant. *27th Symposium on Naval Hydrodynamics*, Seoul, Korea, 5-10 October.
5. Yang, J. et al. (2008). Large-eddy simulation of ship flows with wall-layer models on Cartesian grids. *27th Symposium on Naval Hydrodynamics*, Seoul, Korea, 5-10 October 2008
6. Stern, F. and Agdrup, C. (2008). Proceedings of SIMMAN 2008 workshop on verification and validation of ship maneuvering simulation methods, Lingby, Denmark.
7. Sakamoto, N. (2009). URANS and DES simulations of static and dynamic maneuvering for surface combatant. *Ph.D. thesis in preparation*, The University of Iowa.
8. Yoon, H.S. (2009). Force/Moment and Phase-Averaged Stereo PIV Flow Measurements for Surface Combatant in PMM Maneuvers. *Ph.D. thesis in preparation*, The University of Iowa.
9. Stern, F., Wilson, R. and Shao, J. (2006). Quantitative approach to V&V of CFD simulations and certification of CFD codes. *International Journal of Numerical Methods in Fluids*, **50** (11), 1335-1355.
10. Longo, J., Shao, J., Irvine, M. and Stern, F. (2007). Phase-averaged PIV for the nominal wake of a surface ship in regular head waves. *Journal of Fluids Engineering*, **129** (5), 524-540.

Soroban-Grid CIP Method for Ocean Research and Ship Design - High Performance Computing with Earth Simulator -

Takashi Yabe*, Youichi Ogata, Takeshi Sugimura***, Kenji Takizawa****, and Keiko Takahashi*****

* *Tokyo Institute of Technology, 2-12-1, O-okayama, Meguro, Tokyo, Japan*
(e-mail: yabe@mech.titech.ac.jp)

** *Hiroshima University, 1-4-1, Kagamiyama, Higashi-Hiroshima-shi, Hiroshima, Japan*
(e-mail: yogata@hiroshima-u.ac.jp)

*** *Japan Agency for Marine-Earth Science and Technology, 3173-25, Showa-machi, Kanazawa-ku, Yokohama, Kanagawa, Japan*
(e-mail: sugi@jamstec.go.jp)

**** *Rice University, MS 321 6100 Main Street, Houston, TX 77005, USA*
(e-mail: ktaki@rice.edu)

***** *Japan Agency for Marine-Earth Science and Technology 3173-25, Showa-machi, Kanazawa-ku, Yokohama, Kanagawa, Japan*
(e-mail: takahasi@jamstec.go.jp)

Abstract : We provide an overview of how fluid–solid and fluid–fluid interfaces can be computed successfully with the CIP method[1, 2] based on adaptive Soroban-grid[3], which was combined in for computation of 3D fluid–object and fluid–structure interactions in the presence of free surfaces and fluid–fluid interfaces. This paper shows that the combined technique can be extended to ocean research and ship hydrodynamics, and our objective in this paper is to provide an easy-to-follow description of the key aspects of the approach.
Keywords: Fluid–solid interface, CIP method, Soroban-grid

1. INTRODUCTION

Numerical analysis using Earth Simulator have succeeded in solving very large scale ocean and atmosphere researches. On the other hand, regarding small scale, high performance computing with Earth Simulator will be also very effective in order to deal with fluid–solid interfaces for such as optimization of ship designs.

In recent decades, a substantial number of finite element interface-tracking (moving grid) methods have been developed for computation of fluid–solid interfaces, including fluid–structure interactions (see, for example, [4, 5, 6]).

Accurate methods with more freedom from mesh moving and distortion concerns are always more desirable. A method with such features was recently introduced in [7] by combining the CCUP method [8], which is based on the Constrained Interpolation Profile/Cubic Interpolated Pseudo-particle (CIP) method developed by Yabe et al. [1, 2] for solving hyperbolic equations, and the adaptive "Soroban grid" technique [3], which is an unstructured and collocated grid technique.

Although the solution technique is based on a collocated-grid approach, high-order accuracy and robustness are maintained. The Soroban grid technique does not have any elements or cells connecting the grid points, and therefore the approach is free from mesh (or grid) distortion limitations.

This paper is intended to introduce the Soroban-grid CIP Method for ocean research and ship design. The CIP method and the Soroban grid technique are briefly described in Section 2. The test computations are presented in Section 3 and the concluding remarks are given in Section 4.

2. CIP METHOD AND SOROBAN GRID

The CIP method which has third-order accuracy in time and space is essentially semi-Lagrangian scheme for solving hyperbolic equations, such as:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0. \tag{1}$$

However, unlike conventional semi-Lagrangian schemes, the CIP method uses both values f and spatial derivatives $g = \partial f/\partial x$ to make up cubic-interpolation between two grid points. For example in one-dimension, when f and g are given at two grid points, the profile between these points can be interpolated by cubic polynomial $F(x) = ax^3 + bx^2 + cx + d$. Therefore, the profile at $n+1$ step can be obtained transporting the profile by $u\Delta t$ like $f^{n+1} = F(x - u\Delta t), g^{n+1} = dF(x - u\Delta t)/dx$.

$$a_i = \frac{g_i + g_{iup}}{D^2} + \frac{2(f_i - f_{iup})}{D^3}, \quad b_i = \frac{3(f_{iup} - f_i)}{D^2} - \frac{2g_i + g_{iup}}{D}, \tag{2}$$

$$\begin{aligned} f_i^{n+1} &= a_i X^3 + b_i X^2 + g_i^n X + f_i^n, \\ g_i^{n+1} &= 3a_i X^2 + 2b_i X + g_i^n, \end{aligned} \tag{3}$$

where we define $X = -u\Delta t$ and the supersubscript "n" indicates the time step. Here, $iup(= i - sgn\,(u_i)), sgn\,(u_i) = 1\ (u_i \geq 0), -1\ (u_i < 0), D = -\Delta x \cdot sgn\,(u_i)$. Therefore, the value and its spatial derivative at the next time step $n+1$ are explicitly given. It is possible to extend the scheme to more than one dimension with a directional-splitting or non-directional-splitting technique.

The CIP method has recently been upgraded to include adaptive grids with assurance of both high-order accuracy and robustness. The new grid system is called the Soroban-grid [3]. The schematics of a Soroban grid is shown in Figure 1. The grid system consists of straight lines and grid points moving

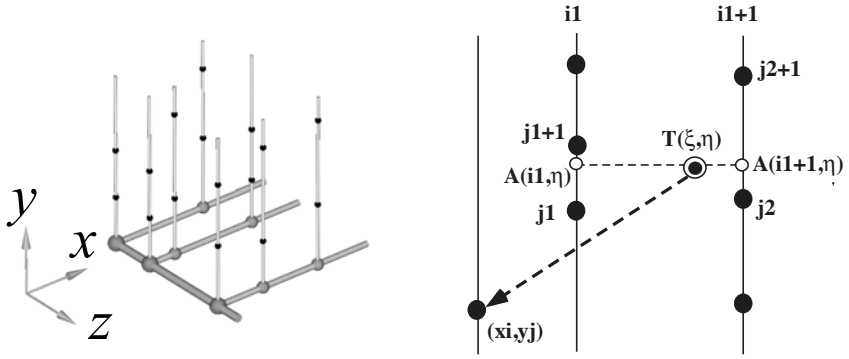


Figure 1: *Soroban grid arrangement. Left: 3D view. Right: view on a plane.*

along those lines, like how it is in an abacus. The lines on a plane move in parallel, and the planes also move in parallel. The length of each line and the number of grid points along each line can be variable.

To understand how advection equations are solved with the Soroban grid, let us consider the grid lines and points on a plane shown in Fig. 1 (Right), where the vertical lines in the y-direction are spaced in the x-direction, and the grid points move along each line. Let (x_i, y_j) be the point of interest. If its upstream departure point T is given as (ξ, η)=$(x_i - u\Delta t, y_j - v\Delta t)$, the solution f^{n+1} of (1) at (x_i, y_j) is simply given by the value at T. As in the CIP formulation in the Cartesian grid, T must be interpolated from the neighboring points.

Since this scheme uses only the one-dimensional CIP method without coordinate transformation, it is able to retain the third-order accuracy in space and time even for deformed grids such as the one demonstrated by Figure 10 in [3].

3. NUMERICAL EXAMPLE

3.1 Incompressible flow past a circular cylinder

First application of the Soroban-grid CIP technique is incompressible flow past a circular cylinder at Reynolds number is 100[9]. The dimensions of the computational domain are 60×16 and the cylinder is located at $(8, 8)$, where the values are normalized by the cylinder diameter. The location of the cylinder and the lateral dimension of the domain are the same as those reported in [10]. The no-slip boundary condition is imposed on the cylinder surface, and inside the cylinder both velocity components are set as $u = v = 0$. The number of grid points is initially about 4,000, eventually increased up to 9,000.

Figure 2 shows an initial grid arrangement and a snapshot of the adaptive grid. It can be seen that Soroban-grid are concentrated on generated Karman vortex. The computed values for the drag and lift coefficients (C_D, C_L) and the Strouhal number are 1.375 ± 0.009, ± 0.27 and 0.16. These values are in good agreement with those reported in [10].

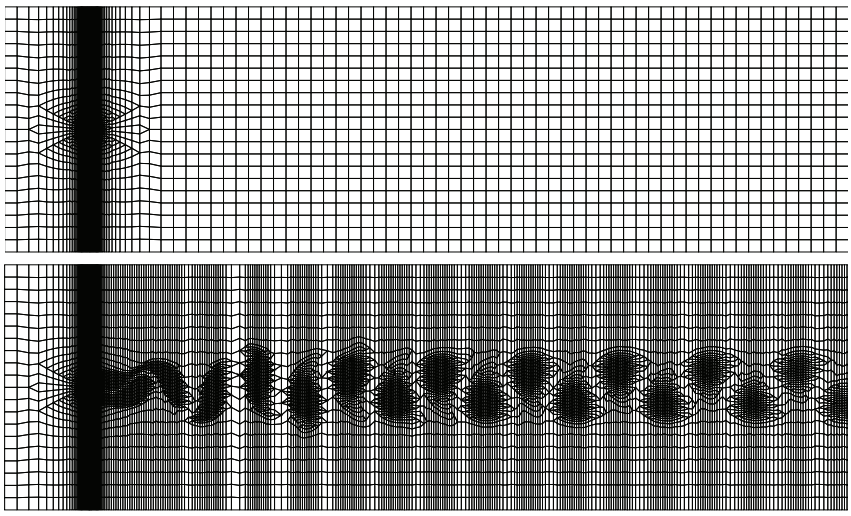


Figure 2: *Flow past a cylinder. The Soroban lines are parallel to the vertical* (y) *axis. Top: initial grid arrangement. Bottom: snapshot of the adaptive grid.*

3.2 Hydrodynamics of a container ship

The ship used in this test computation is modeled after a 5500 TEU container ship and is 284 m long[11]. It is cruising at 10 m/s and is undergoing rigid-body motion with 5 degrees-of-freedom. The water depth is 50 m. The Froude number is 0.19. The computational domain is translating with the ship. The condition we specify at the inflow boundary is the third-order Stokes wave with the wave length and height set at 284 m and 16 m. The lateral boundaries have slip conditions, and the outflow boundary has no condition specified.

The computation is carried out for 100 s. Figures 3 and 4 show the ship and the water surface at $t = 17.9$ s and 24.5 s. Figure 5 shows the diver's view of the ship and the water surface at $t = 24.5$ s, together with the Soroban grid lines and points at that instant.

4. CONCLUSIONS

It is shown by the basic example of flow past a cylinder and the realistic simulation of the hydrodynamics of a container ship in this paper that the CIP-CUP method in combination with the adaptive

Figure 3: *Hydrodynamics of a container ship. The ship and the water surface at* $t = 17.9$ *s.*

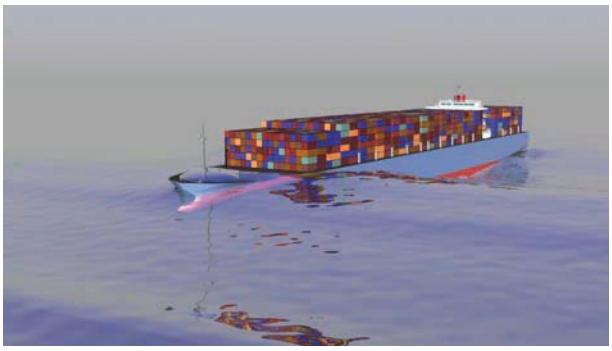


Figure 4: *Hydrodynamics of a container ship. The ship and the water surface at* $t = 24.5$ *s.*

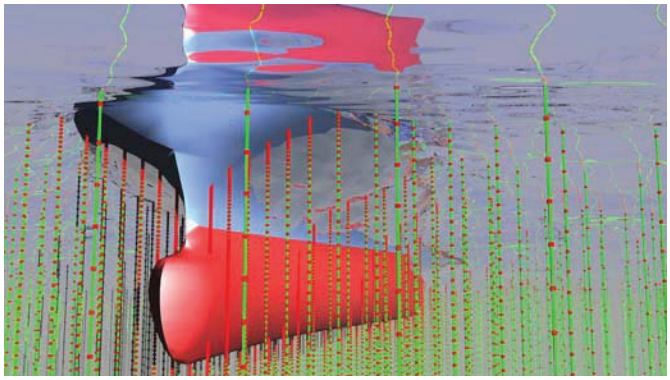


Figure 5: *Hydrodynamics of a container ship. Diver's view of the ship and the water surface at* $t = 24.5$ *s, together with the Soroban grid lines and points at that instant.*

Soroban grid technique can be used for 3D fluid–object and fluid–structure interactions in the presence of free surfaces and fluid–fluid interfaces. The Soroban-grid technique, because of its unstructured nature, brings geometric flexibility and makes it possible to generate suitable grids around complex shapes. Although the solution technique is based on a collocated-grid approach, high-order accuracy and robustness are maintained.

REFERENCE

1. T. Yabe, and T. Aoki, "A universal solver for hyperbolic-equations by cubic-polynomial interpolation. I. One-dimensional solver", *Computer Physics Communications*, **66** (1991) 219–232.

2. T. Yabe, F. Xiao and T. Utsumi , "Constrained interpolation profile method for multiphase analysis", *Journal of Computational Physics*, **169** (2001) 556–593.

3. T. Yabe, H. Mizoe, K. Takizawa, H. Moriki, H. Im, Y. Ogata, "Higher-order schemes with CIP method and adaptive Soroban grid towards mesh-free scheme",*Journal of Computational Physics*, **194** (2004) 57–77.

4. T.E. Tezduyar, M. Behr, and J. Liou, "A new strategy for finite element computations involving moving boundaries and interfaces – The deforming-spatial-domain/space–time procedure: I. The

concept and the preliminary numerical tests", *Computer Methods in Applied Mechanics and Engineering*, **94** (1992) 339–351.

5. T.E. Tezduyar, M. Behr, S. Mittal, and J. Liou, "A new strategy for finite element computations involving moving boundaries and interfaces – The deforming-spatial-domain/space–time procedure: II. Computation of free-surface flows, two-liquid flows, and flows with drifting cylinders", *Computer Methods in Applied Mechanics and Engineering*, **94** (1992) 353–371.

6. T.E. Tezduyar, "Finite element methods for flow problems with moving boundaries and interfaces", *Archives of Computational Methods in Engineering*, **8** (2001) 83–130.

7. K. Takizawa and T. Yabe and Y. Tsugawa and T.E. Tezduyar and H. Mizoe, "Computation of Free-Surface Flows and Fluid-Object Interactions with the CIP Method Based on Adaptive Meshless Soroban Grids", *Computational Mechanics*, published online, July 2006.

8. T. Yabe, P.Y. Wang, "Unified numerical procedure for compressible and incompressible fluid", *Physical Society of Japan, Journal*, **60** (1991) 2105–2108.

9. T.Yabe, K.Takizawa, T.E. Tezduyar, and H-Nam Im, "Computation of fluid-solid and fluid-fluid interfaces with the CIP method based on adaptive Soroban grids : An overview", *Int.J.Num.Methods in Fluids*, **54** (2007) 841–853.

10. T.E. Tezduyar, J. Liou and D.K. Ganjo, "Incompressible flow computations based on the vorticity-stream function and velocity-pressure formulations", *Computers & Structures*, **35** (1990) 445–472.

11. K.Takizawa, K.Tanizawa, T.Yabe and T.E. Tezduyar, "Ship hydrodynamics computations with the CIP method based on adaptive Soroban grids", *Int.J.Num.Methods in Fluids*, **54** (2007) 1011–1019.

CFD ON THE WORLD'S FOUR FASTEST SUPERCOMPUTERS

Adapting the CFDNS Compressible Navier-Stokes Solver to the Roadrunner Hybrid Supercomputer

Jamaludin Mohd-Yusof*, Daniel Livescu, Timothy Kelly*****

**Los Alamos National Laboratory, Los Alamos, NM 87545, USA*
(Tel: 505-665-2870; e-mail: jamal@lanl.gov)
***Los Alamos National Laboratory, Los Alamos, NM 87545, USA*
(e-mail: livescu@lanl.gov)
****Los Alamos National Laboratory, Los Alamos, NM 87545, USA*
(e-mail: tkelley@lanl.gov)

Abstract: We present a brief overview of the Roadrunner hybrid architecture, followed by a summary of the original code to be adapted. The coding challenges of porting the CFDNS compressible Navier-Stokes solver to the hybrid architecture will be discussed, along with the code modification predicated by the serial speedup on the CBE. Performance results from the initial stages, performed on the single-precision development system, through to the current full simulations, which will be ongoing at the time of the talk, will also be presented. We observe that the overall coding effort, and benefits realized, are considerable, but many of the global modifications to the code will be applicable to a variety of future directions in high-performance computing.

Keywords: turbulence, hybrid architecture, petascale, Roadrunner

1. INTRODUCTION

Petascale computing is expected to bring a number of breakthroughs in science. One of the areas most likely to benefit is fluid turbulence, where very large scale computations can provide insight and indicate appropriate modelling paradigms for the routine coarse mesh calculations needed in applications. Rigorous modelling techniques for the unclosed or sub-grid terms are essential to facilitate the use of these low-resolution calculations in science-based prediction. Turbulence and turbulent mixing are essential in many physical applications. Although important progress has been made in recent years in our understanding of turbulence, complete quantification, prediction, simulation, and control still eludes us. The problem is due in large part to the very broad range of dynamically relevant spatio-temporal scales.

Turbulence theory and the subsequent model development rely on experimental or high resolution Direct Numerical Simulations (DNS) data for development and verification and validation. The DNS technique seeks “exact" solutions to the governing equations, so that all relevant scales are accurately solved, using high resolution numerical simulations based on high order accuracy discretization algorithms. DNS are conducted without resorting to subgrid modelling or the introduction of “artificial" numerical dissipation or other algorithm stabilizing schemes. This offers a degree of control and wealth of information pertaining to the physics of turbulent mixing inaccessible in experiments. The DNS generated database can be used to examine the physics of turbulence, both as a test of modelling ideas as well as a benchmark for verification of other (non-DNS) codes. With the recent advances in supercomputing technology and algorithms, it is now possible to perform simulations at Reynolds numbers comparable or even larger than those obtained in large laboratory experiments. In addition, very large DNS are starting to be performed for more complex flows, beyond the few canonical flows which have traditionally been the subject of the largest simulations. For example, our recent 1024^3 simulations of two-component, buoyancy driven turbulence revealed an unexpected asymmetry in the mixing [1] (Figure 1), not captured by current experiments of similar flows due to the low resolution of the density measurements. This asymmetry was also confirmed in a more complex flow, a 3072^3 simulation of the Rayleigh-Taylor instability [2], and may explain the bubble-spike anomaly (higher velocities on the light fluid side compared to the heavy fluid side), which is a long-standing open problem in this flow.

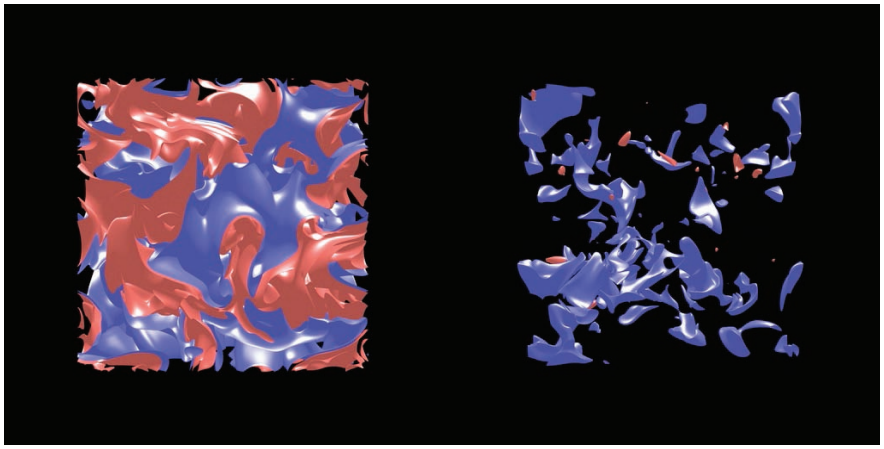


Figure 1: *Mixing between two fluids with different densities. Initially there are equal amounts of the fluids (a). As the mixing proceeds, there is a clear asymmetry between the amount of pure light fluid (red) and the pure heavy fluid (blue) left in the flow (b).*

2. SYSTEM OVERVIEW AND BACKGROUND

Roadrunner uses heterogeneous compute nodes. Each consists of two AMD Opteron dual-core microprocessors, communicating via PCIe connections with two pairs of enhanced double precision Cell microprocessors, or Cell Broadband Engines (CBEs). The four Opteron cores have non-uniform memory access to 16 GB of DDR2 RAM, while each pair of CBE's has non-uniform access to 8 GB DDR2 RAM. The disjoint nature of the memory spaces requires the programmer to explicitly transfer data between them. Notably, only the Opterons have access to the Infiniband cluster interconnect; the CBEs therefore function as accelerators to the Opterons. To date, application codes have maintained the one CBE per Opteron ratio, though this is not required by the system architecture.

Each CBE is itself a heterogeneous system with one PowerPC core (PPE) and 8 Synergistic Processing Element cores (SPEs). The Element Interconnect Bus (EIB) connects the SPEs, the PPE, and Cell main memory; the EIB sustains 25.6 GB/s aggregate bandwidth between SPEs and main memory. The SPEs are simple, low-power, in-order vector cores with large uniform register files. An SPE loads and stores data and instructions from a 6-cycle latency 256KB local store that, while reminiscent of an L2 cache, is not coherent with main memory. Instead, the programmer explicitly moves data between main memory and the SPE's local store using the SPE's Memory Flow Controller (MFC). Each SPE supports 16 memory transfers in flight concurrently.

The full machine has 3060 compute nodes; 180 compute nodes and 12 I/O nodes are joined into a connected unit (CU); 17 CUs are connected by a second switch stage. Roadrunner was the first to break the petaflop/s barrier, and it did so more efficiently than normal at this scale. While Roadrunner is #1 on the June and November, 2008 Top500 lists [3], it is also #7 on the November, 2008 Green500 list [4]—becoming the first #1 Top500 machine to place in the top 10 for efficiency. Roadrunner is cited for "extraordinary energy efficiency... For comparison, the last two supercomputers to top the TOP500 are #43 and #499 on the Green500." [5]

3. CFDNS OVERVIEW

The CFDNS code solves the Navier-Stokes equations (both compressible and incompressible cases) and species transport equations on structured grids. Cartesian, cylindrical, and spherical coordinates are allowed. The spatial derivatives are evaluated using compact (Pade) finite differences, with very low numerical dissipation. Schemes with sixth order accuracy are used in the present version. Compared to central finite differences of the same order, compact schemes provide a better representation of the shorter lengthscales, which is an important aspect in turbulence calculations. This feature brings them closer to spectral methods, traditionally used in DNS of turbulent flows, while maintaining freedom in choosing the mesh geometry and the boundary conditions. Compact schemes of

the same order also require a smaller stencil than central schemes, which improves the accuracy and efficiency of the boundaries treatments. The derivative calculations require solutions of linear tridiagonal systems. The rest of the calculations require point-wise updates of various quantities and are cache friendly and easily parallelized or vectorized. The time advancement is performed using the variable time stepping explicit Runge-Kutta-Fehlberg (RK45) scheme. The boundary conditions can be either periodic or non-reflecting inflow/outflow and/or walls.

4. ROADRUNNER PROGRAMMING CHALLENGES

The doubly heterogeneous computing environment of the Roadrunner system introduces new complexity into the programming model. In practice, most application codes have used the Opterons to implement a standard SPMD MPI program, and then offloaded some significant amount of work to the CBE. The CBE program divides this work into a multithreaded application, with one thread running on each SPE. The PPE coordinates communication between SPE threads and the Opteron, and the Opterons coordinate inter-rank communication, in addition to any other work they may be assigned.

Programming challenges then include developing the three programs required by the Opteron, PPE, and SPEs. Data motion must be programmed explicitly. The programmer employs Cell-specific C language extensions, or intrinsics, in order to use the SPE's memory flow controller and to make best use of its vector instruction set (this last point is no different than SSE programming). The SPE's 256 kB local store requires judicious use, though again, this is not tremendously different from cache-aware programming. However, the changes to underlying data structures and algorithms induced by Roadrunner are shown to be advantageous to all of the current high-performance hybrid and conventional acceleration frameworks (Cell, GPU, SSE) and are therefore fundamentally necessary to leverage future architectures. From this perspective, Roadrunner is a conservative machine, a bridge between yesterday's homogeneous clusters and tomorrow's heterogeneous hardware.

5. HYBRID CODE LAYOUT

Here we discuss the overall layout of the RRDNS code, and show how the choices are dictated by the architecture.

5.1 Division of Work

The bulk of Roadrunner's compute power lies in the Cells, with only ~50 Tflops attributed to the Opterons. This, combined with the need to minimize communication across the PCIe and Infiniband channels, dictates that the application reside primarily on the Cell blades. Similarly, all MPI and storage functions must reside on the Opterons, since the Cells have no direct access to the network fabric. Note that while this requires the conversion of almost all the original code to SPE-specific versions, it also serves to mask the heterogeneity of the architecture to some extent, except where that can be leveraged to improve performance.

5.2 Reorganization of Data Structures

The SPEs are vector processors, operating on 128-bit quadwords as their fundamental unit. While scalar code is supported by the compiler, this may be significantly slower than vector code and should be avoided. The underlying data structure of the original CFDNS code was an 'array of structures' (AOS) type, with all variables at a single grid-point laid out contiguously in memory, which offers some advantages when same operations are applied to all the variables. The internal data structures of the RR code use 'structure of arrays' (SOA), with the (arbitrarily chosen) z-direction contiguous in memory. This provides a natural vectorization of all point-wise operations, which are the bulk of all sweeps through the data. The need for offset (stencil-type) operations is minimized by the use of Pade compact finite difference schemes, which means that all derivatives are precomputed and thus appear only as element-wise multiply-add operations in the update equations. However, this also reduces the arithmetic intensity of the update equation and increases storage requirements. The regular data layout pattern of a structured grid leads to natural shared-memory division of data and work within each cell, with no memory conflicts.

5.3 Reorganization of Communication Patterns

The performance of the CBE produces greater than order-of-magnitude speedup over the Opteron-only local compute performance, resulting in an imbalance between compute and communication compared to an

unaccelerated cluster. As a result, this requires a re-examination of the original communication patterns to absolutely minimize latency and data movement. Additionally, we recognize that algorithmic changes that result in more local computation but less inter-node communication may be advantageous on this architecture, whereas they might not on a conventional cluster. This leads us to totally re-design the tri-diagonal solver algorithms; the new form requires twice as much local computation, but significantly less data movement, and allows significantly better performance of the parallel solver than would otherwise be possible.

6. PERFORMANCE RESULTS

The serial speedup of the Cell version of the code is approximately 50x [6], which is shown to be reasonable when considering the clock speed, parallelism and vectorization afforded by the Cell. Notably, the excellent performance of the individual memory controllers is responsible, since the low arithmetic intensity of the algorithm does not allow the actual compute power of the SPEs to be utilized. This serial speedup prompted us to perform significant modifications to the parallel code design, as indicated above, since the original data movement patterns would have limited parallel speedup to ~ 5x. Following this redesign, the parallel code exhibits overall speedup in the range of 20x compared to the Opteron-only version [7].

7. CONCLUSIONS

The overall effort required to adapt the CFDNS code to the Roadrunner hybrid architecture was considerable, but the benefits in terms of performance improvements are likewise significant. Several of the improvements prompted by this work can be introduced into conventional versions of the code, with varying results, and would likely not have been attempted without the need to completely rewrite the code for the CBE architecture.

In general, porting of more complex codes will likely require significant language and/or compiler improvements to be feasible. However, the changes made to the underlying data structures and algorithms will likely be beneficial on most future architectures and accelerators, due to the trend towards more vector processors (e.g. Cell, GPU, SSE). This commonality means that the changes outlined here will lead to improved code performance on many emerging architectures for the foreseeable future.

References

1. Livescu, D. and Ristorcelli, J.R. (2008) Variable-density mixing in buoyancy-driven turbulence, *J. Fluid Mech*. 605, 145-180.
2. Livescu, D., Ristorcelli, J.R., Gore, R.A., Dean, S.H., Cabot, W.H. and Cook, A.W. (to appear, 2009) High-Reynolds number Rayleigh-Taylor turbulence, *J. Turb.*
3. http://www.top500.org/lists/2008/11
4. http://www.green500.org
5. http://www.green500.org/news/20080630.php
6. Mohd-Yusof, J., Livescu, D. and Petersen, M. R. (2007) Roadrunner: Compressible Turbulence Simulation," *SC07 (International Conference for High Performance Computing, Network, Storage and Analysis),* Reno, NV.
7. Mohd-Yusof, J., Livescu, D., Kelley, T., Petersen, M. R. and Desai, N. (2008) Simulation of Compressible Fluid Flow on Roadrunner, *SC08 (International Conference for High Performance Computing, Network, Storage and Analysis),* Austin, TX.

Large Eddy Simulation of Turbulence-Chemistry Interactions in Reacting Flows: Experiences on the ORNL NCCS Cray-XT Platforms (Jaguar)

Joseph C. Oefelein[†] and Ramanan Sankaran[‡]

[†]*Combustion Research Facility, Sandia National Laboratories, Livermore, CA 94551, USA*
(Tel: 925-294-2648; e-mail: oefelei@sandia.gov)
[‡]*National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA*
(e-mail: sankaranr@ornl.gov)

Abstract: This paper provides a perspective on high-performance computing as an enabler for high-fidelity simulations of turbulent combustion processes typically encountered in a variety of propulsion and power systems. We focus on recent experiences on the Oak Ridge National Laboratory (ORNL) National Center for Computational Sciences (NCCS) Cray-XT Platforms (i.e., Jaguar) using a unique massively-parallel flow-solver designed for application of the Large Eddy Simulation technique called RAPTOR.

Keywords: Large Eddy Simulation, Turbulent Combustion, Cray-XT Platforms.

1. INTRODUCTION

Application of the Large Eddy Simulation (LES) technique provides the formal ability to treat the full range of multidimensional time and length scales that exist in turbulent reacting flows in a computationally feasible manner. The large energetic-scales are resolved directly. The small subgrid-scales are modeled. This allows simulation of the complex multiple-time multiple-length scale coupling between processes in a time-accurate manner. The combination of LES, high-performance massively-parallel computing, and advanced experiments in combustion science, offer unprecedented opportunities for synergistic, high-fidelity investigations aimed at the development of accurate predictive models. Here we describe what the science drivers are, the related computational demands, and present a set of recent examples. Emphasis is placed on the performance achieved, significant solution results produced as a consequence, and the combined computational effectiveness of the NCCS Cray-XT platforms (Jaguar(pf)) and our flow solver RAPTOR.

1.1. Science Drivers

Turbulent combustion processes are prevalent in a wide variety of propulsion and power systems including internal-combustion (IC) engines, gas-turbines and liquid-rockets. As such, development and rigorous validation of science-based predictive models for turbulent combustion is recognized as an important priority in research and there are a variety of challenges. Turbulent flows involving heterogeneous chemically reacting mixtures (as is the case for all propulsion and power systems) have a variety of complicating factors including highly nonlinear chemical kinetics, small-scale velocity and scalar-mixing, turbulence-chemistry interactions, compressibility effects (volumetric changes induced by changes in pressure), and variable inertia effects (volumetric changes induced by variable composition or heat addition). Coupling between processes occurs over a wide range of time and length scales, many being smaller than can be resolved in a numerically feasible manner. Further complications arise when multiple phases are present due to the introduction of dynamically evolving interface boundaries and the complex exchange processes that occur as a consequence. At the device level, high-performance, dynamic stability, low pollutant emissions, and low soot formation must be achieved simultaneously in highly confined geometries that generate extremely complex flow and acoustic patterns. Flow and combustion processes are highly turbulent (i.e., integral-scale Reynolds numbers of $\mathcal{O}(10^5)$ or greater), and the turbulence dynamics are inherently dominated by geometry or various operating transients. In many cases operating pressures approach or exceed the thermodynamic critical pressure of the fuel or oxidizer.

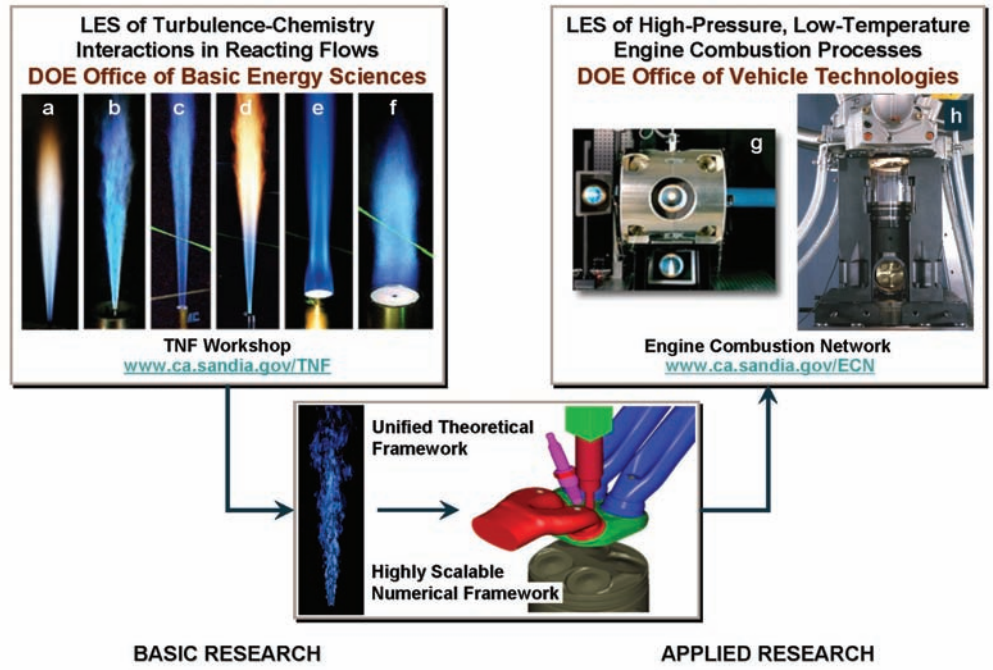


Figure 1: Key experiments currently being studied using RAPTOR. A subset of experiments associated with the Reacting Flow Research program are shown on the left (a,b: Simple jet flames, c,d: Piloted jet flames, e: Bluff-body; f: Bluff-body with swirl). A subset of experiments associated with the Advanced Engine Combustion program are shown on the right (g: Constant-volume Diesel combustion facility, h: Typical single-cylinder optically accessible IC-Engine).

Operation at elevated pressures significantly increases the system Reynolds number(s) and inherently broadens the range of spatial and temporal turbulence scales over which interactions occur.

The limitations and challenges associated with turbulent combustion research requires that a hierarchy of approaches be taken to fully understand key processes and work toward predictive models. The primary challenge is to bridge the gap between basic research and the conditions of interest in typical applications. As part of the Reacting Flow Research and Advanced Engine Combustion programs at Sandia National Laboratories (SNL) Combustion Research Facility (CRF), two complementary projects have been established to achieve this goal. The first is funded under the DOE Office of Science (OS), Basic Energy Sciences (BES) program, and focuses on LES of turbulence-chemistry interactions in reacting multiphase flows. The second is funded under the DOE Office of Energy Efficiency and Renewable Energy (EERE), Office of Vehicle Technologies (OVT) program, and focuses on the application of LES to high-pressure, low-temperature, IC-engine combustion research. Figure 1 shows key experiments currently being studied under these two projects. A subset of experiments associated with the Reacting Flow Research program are shown on the left. A subset of experiments associated with the Advanced Engine Combustion program are shown on the right. Objectives and milestones for both projects are aimed at establishing high-fidelity computational benchmarks that identically match the geometry and operating conditions of key target experiments using a single unified theoretical-numerical framework (i.e., RAPTOR).

Flames studied under Reacting Flow Research (see a-f in Fig. 1 for example) are internationally recognized benchmarks that provide some of the most detailed experimental data available for model validation. Using these data, significant collaborations with key modeling groups worldwide have been established as part of the *International Workshop on Measurement and Computation of Turbulent Nonpremixed Flames* (see Barlow *et al*. [1] for details). The "TNF Workshop" is an ongoing collaboration among experimental and computational researchers. A central theme of the series has been to use detailed comparisons of results from experiments and multiple modeling approaches to quantify state-of-the-art capabilities and identify future research needs toward predictive simulations. As part of this activity, RAPTOR has been used to provide benchmark simulations that reach beyond the capabilities and resources of most universities and industry in a manner consistent with a National Laboratory's role of using high-performance computing. In contrast to the TNF flames, research activities related to Advanced Engine Combustion are focused on IC-engines. Needs and milestones related to RAPTOR have been established in three critical areas: 1) perform

a progression of LES studies focused on the CRF optically accessible hydrogen-fueled IC-engine (see h in Fig. 1), 2) establish a parallel task focused on homogeneous-charged compression-ignition (HCCI) engines, and 3) perform a series of supporting studies focused on the development and validation of multiphase injection and combustion models with emphasis placed on direct-injection processes in IC-engines (see g in Fig. 1). The integrated set of research includes an optimal combination of in-cylinder and canonical (out-of-engine) studies to validate and understand key phenomenological processes that are present in IC-engine flow environments.

Milestones associated with the cases shown in Fig. 1 are all "grand-challenge" in nature and require significant amounts of CPU time (i.e., approximately 1 to 5-million cumulative CPU hours per case). In general, three distinct technical capabilities are required simultaneously: 1) an established theoretical-numerical framework, 2) specialized massively-parallel software that scales efficiently on $\mathcal{O}(10^4 - 10^5)$ processors, and 3) access to "capability-class" computers (which, as defined by the DOE, are platforms designed to provide a small number of users large amounts of CPU time to perform heroic "grand-challenge" calculations). Items 1 and 2 are facilitated by our flow solver, RAPTOR. Item 3 is facilitated in collaboration with Oak Ridge National Laboratory as part of the 2009 INCITE project entitled *High-Fidelity Simulations for Clean and Efficient Combustion for Alternative Fuels*. Key details related to RAPTOR are listed below. Details associated with the ORNL Cray-XT platforms can be found at www.nccs.gov.

1.2. Theoretical-Numerical Framework

RAPTOR is a massively parallel flow solver designed specifically for application of the Large Eddy Simulation (LES) technique to turbulent, chemically reacting, multiphase flows. It solves the fully coupled conservation equations of mass, momentum, total-energy, and species for a chemically reacting flow system (gas or liquid) in complex geometries. It also accounts for detailed chemistry, thermodynamics, and transport processes at the molecular level and uses detailed chemical mechanisms. The code is sophisticated in its ability to handle complex geometries and a generalized subgrid-scale model framework. It is capable of treating spray combustion processes and multiphase flows using a Lagrangian-Eulerian formulation. The numerical formulation treats the compressible form of the conservation equations, but can be evaluated in the incompressible limit. The theoretical framework handles both multi-component and mixture-averaged systems. The baseline formulation also employs a general treatment of the equation of state, thermodynamics, and transport properties that accommodates real gas or liquids with detailed chemistry (i.e., not constrained to ideal gas applications). Details are given by Oefelein [2].

The temporal integration scheme employs an all Mach number formulation using the dual-time stepping technique with generalized preconditioning. The approach is fourth-order accurate in time and provides a fully-implicit solution using a fully explicit (highly-scalable) multistage scheme in pseudo-time. Preconditioning is applied on an inner pseudo-time loop and coupled to local time-stepping techniques to minimize convective, diffusive, geometric, and source term anomalies (i.e., stiffness) in an optimal manner. The spatial scheme is designed using non-dissipative, discretely-conservative, staggered, finite-volume differencing. The discretization is formulated in generalized curvilinear (i.e., body-fitted) coordinates and employs a general R-refinement adaptive mesh (AMR) capability. This allows us to account for the inherent effects of geometry on turbulence over the full range of relevant scales while significantly reducing the total number of grid cells required in the computational domain. Treating the full range of scales is a critical requirement since turbulence-chemistry interactions are inherently coupled through a cascade of nonlinear interactions between the largest and smallest scales of the flow. The second-order accurate staggered grid formulation (where we store scalar values at cell centers and velocity components at respective cell faces) fulfills two key accuracy requirements. First, it is spatially non-dissipative, which eliminates numerical contamination of the subgrid-scale models due to artificial dissipation. Second, the stencils provide discrete conservation of mass, momentum, total energy and species, which is an imperative requirement for LES. This eliminates the artificial build up of velocity and scalar energy at the high wavenumbers, which causes both accuracy problems and numerical instabilities in turbulent flow calculations.

The code framework is massively-parallel and has been optimized to provide excellent parallel scalability attributes using a distributed multiblock domain decomposition with generalized connectivity. Distributed-memory message-passing is performed using MPI and the Single-Program–Multiple-Data (SPMD) model. It accommodates complex geometric features and time varying meshes with generalized hexahedral cells while maintaining the high accuracy attributes of structured spatial stencils. The numerical framework has been ported to all major platforms and provides highly efficient coarse- and fine-grain (i.e., weak and strong) scalability attributes. The code is fully vectorized and has been optimized for both vector and commodity architectures. Further optimization is currently in progress to account for new issues associated with state-of-the-art multi-core technology. The complete package is fully modular, self-

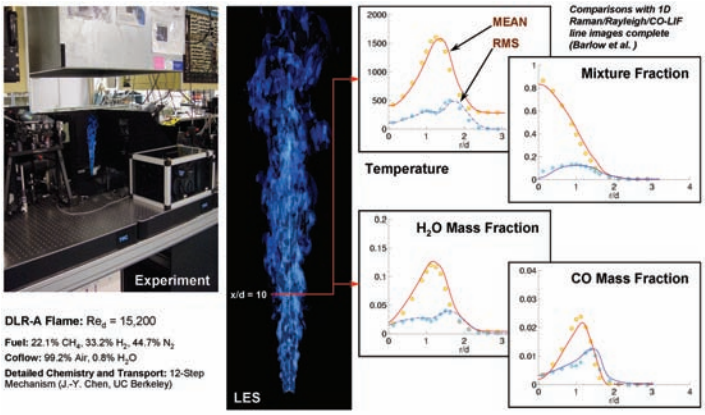


Figure 2: To the left is a photograph of the DLR-A flame in the experimental test section (corresponds to b in Fig. 1). At center is the corresponding solution from LES. To the right are representative comparisons between experimentally measured (symbols) and modeled (lines) results showing acceptable agreement.

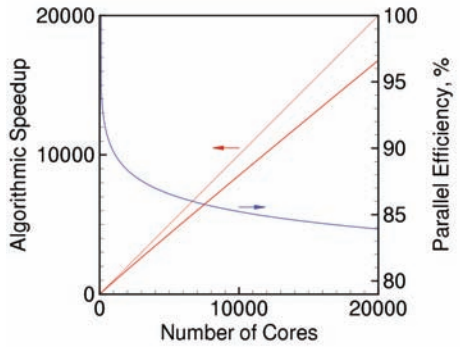


Figure 3: Strong (fine-grain) scaling attributes exhibited by RAPTOR on the ORNL NCCS CRAY-XT4 (Jaguar).

contained, and written in ANSI standard Fortran 90. The complete theoretical-numerical framework (i.e., governing equations, physical submodels, numerics and , parallel efficiency) has been extensively validated over the last 17 years. Representative results can be found in Refs. [3–7].

2. REPRESENTATIVE RESULTS

Porting the RAPTOR software framework to the NCCS Cray-XT platforms (Jaguar(pf)) was fairly routine with no major problems. A representative set of results are shown in Figs. 2 and 3. To the left in Fig. 2 is a photograph of the DLR-A flame in the experimental test section at Sandia. This flame corresponds to b in Fig. 1. At center is the corresponding solution from LES, which was implemented by gridding the entire experimental test section. The total grid size was 10,285,056 computational cells. The plots to the right in Fig. 2 show comparisons between numerical results via RAPTOR (lines) and measured Raman/Rayleigh/CO-LIF line image data (symbols). Here we show mean and RMS profiles. These results, coupled with similar comparisons performed throughout the domain, provide a validated level of confidence in the accuracy of the solution.

Using the case shown above, the computational performance was evaluated in two stages. First, we performed a series of strong and weak scaling studies to determine the baseline performance for both modes of operation. Second, we benchmarked the serial performance of the code to establish an initial baseline with respect to performance on a per core basis (i.e., percent of peak). Figure 3 shows a representative set of strong (fine-grain) scaling attributes exhibited by RAPTOR on up to 20,000 processor cores. Results were obtained by holding the total grid size fixed and successively increasing the number of processors used to perform the calculation. Note that this particular case produces an extremely fine-grain test where the percent of communicated grid cells per processor was approaching 40 percent of the total. Maintaining the level of performance indicated for such a fine-grain decomposition can be attributed to the explicit nature of the solver. Our dual-time stepping methodology provides the fully-coupled implicit solution to the governing equations, but achieves this by iterating within an inner pseudo-time loop using an explicit multistage scheme.

More recently, we have initiated a series of weak scaling studies. The computational performance was simultaneously evaluated by using a CrayPAT instrumented executable in place of the original. The program was instrumented to provide hardware performance counter information from start to finish. The simulation was performed using 47,616 processor cores on the Cray XT5 system. The CrayPAT output was postprocessed using pat_report. On average, each processor core performed 7.94-billion floating point operations leading to an aggregate of 378-trillion floating point operations being performed by the 47,616 cores. We measure the computational performance of RAPTOR for a given problem using the metric

$$\text{Performance} \;=\; \text{CPU time} \;/\; \text{Number of Grid cells} \;/\; \text{Number of time-steps},$$

where CPU time is the product of two quantities: (i) wall-clock time taken from start to finish of the time integrator

portion of the solver, and (ii) the number of processor cores occupied by the job while the program was executing. For the benchmark run, the code's internal timers reported that the time integration through 50 time-steps took 1034 seconds. The remaining time (approximately 300 seconds) was consumed by the initialization step when the computational mesh and initial condition information were read from the disk and the software prepared itself for the simulation. Therefore, the performance of RAPTOR during the benchmark simulation was

$$(1,034 \text{ seconds } \times \ 47,616 \text{ cores}) \ / \ 10,285,056 \text{ cells} \ / \ 50 \text{ time-steps } = \ 0.096.$$

It cost 96-milliseconds of processor time per cell per time-step to simulate the problem on 47,616 cores. Subsequent runs are being performed by systematically increasing the total CPU time required (i.e., total number of floating point operations per case) by factors of 2 as a function of increasing jet Reynolds number. Our goal is to demonstrate that we can simulate successively larger problems in the same amount of time on this platform. Details related to these studies will be provided in the paper.

REFERENCES

1. R. S. Barlow. International workshop on measurement and computation of turbulent nonpremixed flames. www.ca.sandia.gov/TNF, 1996-2009. Combustion Research Facility, Sandia National Laboratories.

2. J. C. Oefelein. Large eddy simulation of turbulent combustion processes in propulsion and power systems. *Progress in Aerospace Sciences*, 42(1):2–37, 2006.

3. J. C. Oefelein. Thermophysical characteristics of LOX–H_2 flames at supercritical pressure. *Proceedings of the Combustion Institute*, 30:2929–2937, 2005.

4. J. C. Oefelein. Mixing and combustion of cryogenic oxygen-hydrogen shear-coaxial jet flames at supercritical pressure. *Combustion Science and Technology*, 178(1-3):229–252, 2006.

5. J. C. Oefelein, R. W. Schefer, and R. W. Barlow. Toward validation of LES for turbulent combustion. *AIAA Journal*, 44(3): 418–433, 2006.

6. J. C. Oefelein, V. Sankaran, and T. G. Drozda. Large eddy simulation of swirling particle-laden flow in a model axisymmetric combustor. *Proceedings of the Combustion Institute*, 31:2291–2299, 2007.

7. T. G. Drozda and J. C. Oefelein. Large eddy simulation of direct injection processes for hydrogen and LTC engine applications. *SAE World Congress, Paper 2008-01-0939*, April 14-17 2008. Detroit, Michigan.

Large Scale Aerodynamic Calculation on Pleiades

Thomas H. Pulliam, Dennis C. Jespersen

NASA/Ames Research Center, Moffett Field, CA 94035, USA
e-mail: Thomas.H.Pulliam@nasa.gov, Dennis.Jespersen@nasa.gov

Abstract: We present a very large scale aerodynamic calculation on the NASA Pleiades supercomputer using the three-dimensional Navier-Stokes code OVERFLOW. The application is a rotorcraft blade simulation, with attention focused on the resolution of the vortices which develop off the blade tips and propagate through the flow domain. This simulation represents a level of refinement beyond any previous results presented for this problem. This extremely large CFD application stresses the capabilities of the Pleiades system, provides a relevant application with high levels of grid resolution providing benchmark results, and helps to assess the capabilities of the OVERFLOW code in terms of scalability, parallel efficiency and performance.
Keywords: large scale aerodynamics, supercomputer, rotorcraft

1. INTRODUCTION

A very large scale aerodynamic calculation on the NASA Pleiades supercomputer is presented here. There are three main purposes for this effort. First, we define an extremely large CFD application to stress the capabilities of the Pleiades system in terms of job size, file size, memory use, disk traffic, message-passing, and post-processing of the extremely large data sets produced. Second, a relevant application was chosen which at high levels of grid resolution would provide benchmark results for comparison with coarser grid results and with new grid adaption techniques. Third, we assess the capabilities of the OVERFLOW code in terms of scalability, parallel efficiency and performance under the stress of an extremely large problem. This work emphasizes the capability of the Pleiades system to perform very large aerodynamic calculations.

The application is a rotorcraft blade simulation, which involves complicated physics and geometry, and is one of the more challenging physical applications in terms of CFD resources. In particular, we focus on the resolution of the vortices which develop off the blade tips and propagate through the flow domain. Insufficient resolution causes the vortices to decay too rapidly and to convect to improper positions. This simulation represents a degree of refinement beyond any previous results presented for this problem.

2. PLEIADES ARCHITECTURE

The Pleiades supercomputer at NASA/Ames Research Center is an SGI Altix ICE system with 5888 nodes and 8 cores per node, each node consisting of dual quad-core Intel Xeon EX5472 ("Harpertown") processors for a total of 47104 cores. Each quad-core chip has a 6 MB L2 cache which is shared among the cores on that chip. The cores have a clock frequency of 3.0 GHz with a maximum of 4 floating-point operations per clock per core. The front-side bus frequency is 1600 MHz with a peak transfer rate of 12.8 GB/sec. Most of the nodes have 1 GB of memory per core, but there are 65 "bigmem" nodes that have 2 GB of memory per core.

The nodes are connected by two InfiniBand fabrics, each fabric supporting both message-passing and I/O traffic. At the time of our tests, all large-scale jobs on the system shared a single Lustre filesystem. Jobs are run in a batch environment with scheduling by the Portable Batch Scheduler (PBS).

3. THE OVERFLOW CODE

OVERFLOW [2,5] solves the Reynolds-averaged Navier-Stokes (RANS) equations in generalized coordinates on a system of overset grids. The code has options for several solution algorithms; in this work we used third-order central differencing with artificial dissipation terms in space and the diagonalized Beam-Warming implicit three-factor central-difference scheme for time-stepping [6]. The solutions presented here were run in steady-state mode (time accurate capability is available). The code has algebraic, one-equation, and two-equation turbulence models.

The overset grid approach was originated [1] to simplify the spatial discretization of complex geometries using structured grids. A complex geometry is subdivided into a set of components with relatively simple geometry, and an appropriate grid placed around each component. The component grids are overlaid, or overset, to form a composite grid. Trilinear interpolation is used to transfer computed data between grids. Domain connectivity is accomplished using an object X-ray and automatic hole cutting technique, see [3,4].

Parallelism in OVERFLOW can occur at two levels. At the higher level there is explicit-message passing using MPI. There is an optional lower level of parallelism using OpenMP, which needs shared memory. To give good load balance, OVERFLOW will split grids which contain more than some target number of grid points. This target can be user-specified, or if not specified the code chooses half the average number of grid points per MPI rank; in practice this almost always provides good load balance. Each MPI rank is eventually assigned one or more grids (for large-scale problems with good load balance, there are typically 2 to 3 grids per MPI rank). The code then proceeds to march in time, with each time step having two stages. The first stage is communication: each MPI rank sends all the inter-grid data needed by other MPI ranks, and receives all necessary data. The second stage is computation: each MPI rank computes on its grids, with no message-passing involved. After the second stage all MPI ranks send a small amount of information (residuals, minimum pressure and density, etc.) to the MPI rank 0 process. All I/O is via the MPI rank 0 process.

Most OVERFLOW users utilize only the MPI level of parallelism. This is sufficient for problems which use up to a few hundred or so MPI ranks. For very large scale computations with thousands of MPI ranks, using only the outer level of parallelism would result in a tremendous amount of grid splitting with possibly a significant increase in the number of grid points due to "ghost" points generated during splitting. This increase in grid points would result in increased computation time and increased message-passing during the communication phase of the code; it also would increase memory use. During preliminary work on this project we encountered situations where a given number of cores in pure MPI mode were unable to run a case due to insufficient memory, while the same number of cores in hybrid parallel mode (MPI for outer parallelism among nodes, OpenMP for inner parallelism within a node) could run the case successfully. In the computation presented here we used hybrid mode, with MPI for parallelism at the outer level, among nodes, and OpenMP for parallelism at the inner level, within a node.

4. PHYSICAL PROBLEM

The Tilt Rotor Aeroacoustics Model (TRAM), a 1/4-scale three blades and hub component of the V-22 Osprey tiltrotor aircraft, was chosen for this study. Extensive details of the physical problem, geometry, experimental data and previous numerical studies can be found in Potsdam and Strawn [8]. The goal is to compute the steady hover mode of the TRAM blade system. Therefore, the OVERFLOW computations were performed in a steady non-inertial frame where the observer is moving with the blades and the flow appears stationary with respect to the observer. The one-equation Baldwin-Barth turbulence model was used with corrections, Potsdam and Pulliam [7].

The specifics of the grid system which are pertinent to this study are briefly discussed here. The three-bladed TRAM rotor system is shown in Figure 1, along with a slice of the off-body grid system. In terms of the unstructured overset methodology employed by OVERFLOW, near-body (NB) curvilinear structured grids are generated about the blade and hub geometries with sufficent resolution to capture viscous effects. The reference length scale for this problem is the tip chord (C_{tip}) of the blades. Off-body Cartesian grids are automatically generated by OVERFLOW. The "Baseline" case as defined in Potsdam and Strawn uses a Level 1 grid (L_1) which is uniform in all coordinate directions with a grid spacing $\Delta L_1 = 0.1 C_{\text{tip}}$. The near-body grids are completely embedded within the L_1 grid, where the overset methodology of hole-cutting, iblanking, and Chimera interpolation logic are employed to interface the grid systems [3]. Subsequent off-body Level 2 and higher Cartesian brick grids are generated with spacing $\Delta L_i = 2^{i-1} \Delta L_1$ to extend the grid to the outer boundary of the computational domain, see Figure 1.

The Baseline grid of Potsdam and Strawn is a standard for comparison and will be the basis for our comparisons here. Table 1 lists the characteristics of the Baseline grid and a sequence of refined grids. These grids are a sequence of halvings of the L_1 spacing to improve the resolution of the wake vortices being shed off the blade tips. The "Huge" grid point mesh is the one chosen for the current study which represents three

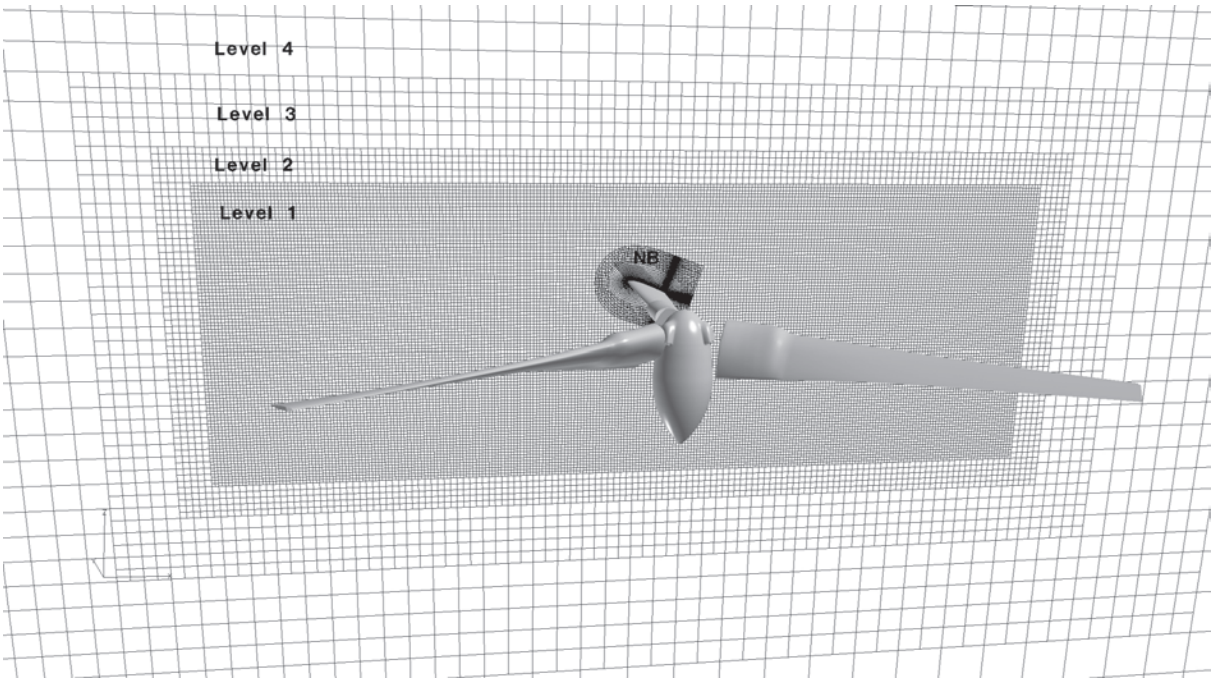


Figure 1: TRAM V22 geometry, Hover

levels of refinement of the L_1 spacing and produces a grid with over 3 billion points, possibly the largest aerodynamic RANS CFD calculation to date.

5. RESULTS

Shown in Figure 2 is a comparison of the solution on the Baseline grid system with the solution on the refined Huge grid system. The results of the Baseline case are exactly consistent with those reported by Potsdam and Strawn [8] and Potsdam and Pulliam [7]. In Figure 2, iso-surfaces of vorticity show that the refined grid captures the wake vortex more accurately than the Baseline grid, producing significantly less decay of the vortex cores. In the Baseline calculation at the contour level chosen, the vortices dissipate after turning approximately 90 degrees from the blade tip. In the Huge case, at the same contour level, the vortices persist for more than 360 degrees before dissipating. Also shown in the Huge case is the ability to capture fine details of the shear layers shed from the blades and the interaction of the shear layers with the tip vortices.

6. OBSERVATIONS

Table 2 shows some performance data for OVERFLOW on the Huge case. In the table under Layout, "$m \times n$" means m MPI ranks with each MPI rank having n OpenMP threads. The grid has 194 zones with 3,087,812,624 total points.

Performance is in terms of nanosec/pts/timestep, so lower is better. This metric is computed both in terms of the nominal number of grid points (3087812624) and the actual number of grid points after splitting. The

Case	ΔL_1	No. of grid Points	No. of grids
Baseline	$0.1 \cdot C_{\text{tip}}$	13,692,987	50
Medium	$0.05 \cdot C_{\text{tip}}$	58,426,144	70
Large	$0.025 \cdot C_{\text{tip}}$	379,761,734	188
Huge	$0.0125 \cdot C_{\text{tip}}$	3,087,812,624	194

Table 1: Grid size for different refinement levels

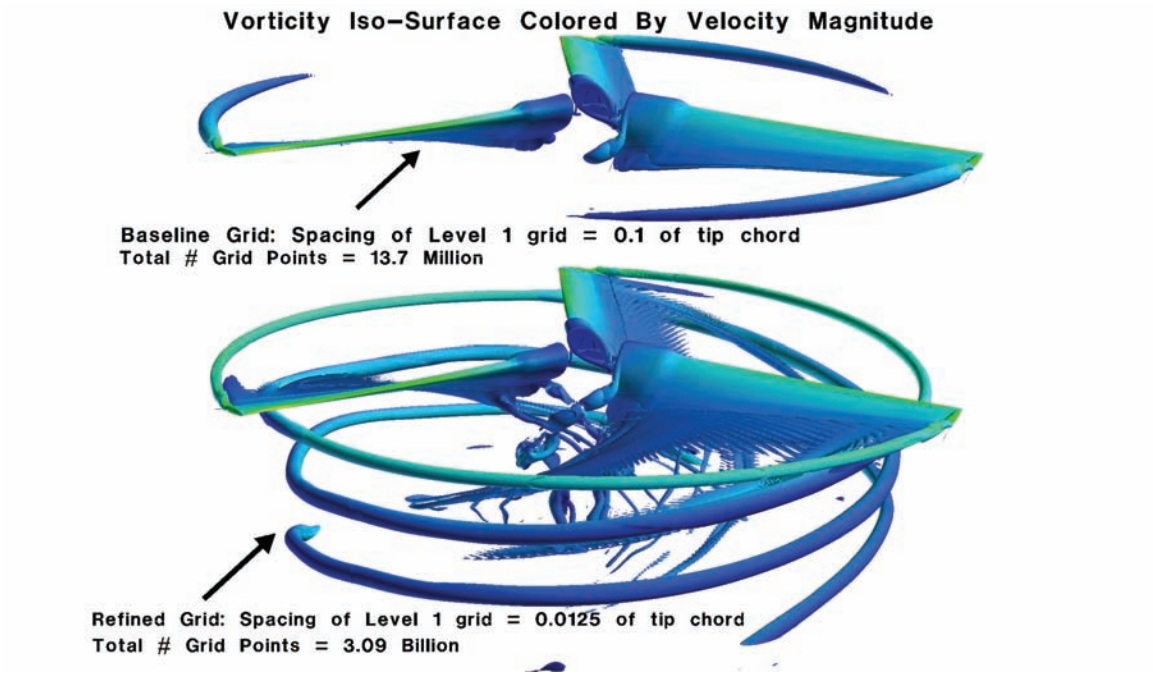


Figure 2: TRAM Comparison

	after splitting		nanosec/pt/timestep	
Layout	no. zones	no. pts	nominal pts	actual pts
128 × 8	322	3129428624	N/A	N/A
192 × 8	580	3189903794	8.815	8.533
256 × 8	744	3209504544	7.047	6.780
512 × 8	1565	3290837114	2.951	2.769
1024 × 8	3227	3405635864	1.242	1.126
2048 × 8	6880	3550830749	0.574	0.499

Table 2: OVERFLOW performance for Huge case

128×8 case did not run due to insufficient memory. All these runs were made in a normal shared environment (not in dedicated time), and all were made with the MPI rank 0 process on a "bigmem" node. Notice the increase in number of zones and number of grid points with the increasing number of MPI ranks. Also notice the good scaling of the code as the number of MPI ranks is increased while the number of OpenMP threads is fixed. The observed superlinear convergence is probably due to cache effects, with more total cache memory available at higher node counts. Message-passing takes a small portion of total time for all these cases.

The result shown in section 5 was computed with the 2048 × 8 processor layout and was run for 40000 time steps. Walltime per step was about 1.77 seconds; at this computational rate 40000 steps took about 20 wallclock hours (spread over several runs). This does not include startup, shutdown, and checkpoint time which added about another 4 wallclock hours. Time to write a solution (including sending all data to MPI rank 0) was in the 16-17 minute range. We realize the bottleneck from routing all I/O through one MPI rank and are vigorously investigating forms of parallel I/O.

Some early attempts to run cases with 16384 cores ran afoul of various problems. A common problem was difficulty in the software underlying the MPI library, which was either nonoperational or appeared to be nonoperational, causing the job to fail to start. A particular issue with large node-count jobs is the increased possibility of single-point hardware problems, causing the whole job to fail.

The solution file for the Huge case is 173 GB in size and the grid file is 86 GB. Disk space restrictions

made it important to quickly copy data off Pleiades, but at the time these computations were performed the maximum data transfer rate to archival storage was about 115 MB/sec, and often rates an order of magnitude less were seen. This rate made expeditious transfer of large files tedious.

This work illustrates the capability of the Pleiades system to handle very large aerodynamic calculations. The solution was obtained in less than 24 hours of total wallclock time, which for a problem of this size is a significant achievement.

REFERENCES

1. Benek, J. A., Buning, P. G., and Steger, J. L. (1985). A 3-D chimera grid embedding technique, *AIAA 7th Computational Fluid Dynamics Conference*, Cincinnati, OH, paper no. AIAA-85-1523.
2. Buning, P. G., Chiu. I. T., Obayashi, S., Rizk, Y. M., and Steger, J. L. (1988). Numerical simulation of the integrated space shuttle vehicle in ascent, *AIAA Atmospheric Flight Mechanics Conference*, Minneapolis, MN, paper no. AIAA-1988-4359.
3. Chan, W. M. (2009). Overset grid technology development at NASA Ames Research Center. *Computers and Fluids*, Vol. 38, No. 3, pp. 496–503.
4. Meakin, R. L. (2001). Automatic off-body grid generation for domains of arbitrary size, *AIAA 15th Computational Fluid Dynamics Conference*, Anaheim, CA, paper no. AIAA-2001-2536.
5. Nichols, R. H., Tramel, R. W., and Buning, P. G. (2006). Solver and turbulence model upgrades to OVERFLOW 2 for unsteady and high-speed applications, *AIAA 24th Applied Aerodynamics Conference*, San Francisco, CA, paper no. AIAA-2006-2824.
6. Pulliam, T. H. and Chaussee, D. S. (1981). A diagonalized form of an implicit approximate factorization algorithm, *J. Comp. Phys.*, Vol. 39, pp. 347–363.
7. Potsdam, M. A. and Pulliam, T. H. (2008). Turbulence Modeling Treatment for Rotocraft Wakes, *AHS Aeromechanics Specialist's Meeting*, San Francisco, CA.
8. Potsdam, M. A. and Strawn, R. C. (2005). CFD simulations of tiltrotor configurations in hover, *Journal of the American Helicopter Society*, Vol. 50, No. 1, pp. 82–94.

On the Performance of the Miranda CFD Code on Multicore Architectures

**Martin Schulz, Andrew W. Cook, William H. Cabot,
Bronis R. de Supinski, William D. Krauss**

Lawrence Livermore National Laboratory, Livermore, CA, USA
(e-mail: {schulzm,awcook,cabot1,bronis,krauss}@llnl.gov)

Abstract: We are currently seeing a trend towards multi- and many-core based parallel systems. In order to exploit such architectures efficiently, applications must make use of all cores within a CPU and consequently have to be able to deal with additional problems caused by this trend, including shared caches, increased memory contention and limited off-chip bandwidth. In this work we investigate the impact of such architectures on the parallel CFD code Miranda and present preliminary results. We study the code on several current multi-core systems and show that, while not a problem just yet, future systems may pose substantial hurdles.
Keywords: Miranda, Fluid Dynamics, Performance Evaluation, Multicore

1 INTRODUCTION

Miranda is a high order 3D hydrodynamics code for computing fluid instabilities and turbulent mixing. It is primarily used to study Rayleigh-Taylor (R-T) and Richtmyer-Meshkov (R-M) instabilities, which occur in supernovae and inertial confinement fusion. Figure 1 shows a sample result from a simulation of a turbulent state of a Rayleigh-Taylor instability. The application uses a massively parallel spectral/compact solver for variable-density incompressible flow, including viscosity and species diffusivity effects [2, 1].

Miranda has been run successfully on a variety of machines, including LLNL's large scale Blue Gene/L (BG/L) system [3]. Figure 2 shows the speedup results of a scaling study on up to 65,536 nodes. Weak scaling results (fixed workload per node) are shown on the left side of Figure 2 and strong scaling results (fixed problem size) on the right. The weak scaling results show that Miranda yields near perfect scaling on the BG/L architecture. Furthermore, the numbers for strong scaling show a superlinear speedup for up to 32K nodes and only a slight degradation on 64K nodes.

However, future architectures will impose new challenges on the scaling behavior of CFD codes, such as Miranda. In particular, the trend towards many- and multi-core chips will have a significant impact on codes' performance. LLNL's systems already reflect this trend: Blue Gene/L (BG/L) was limited to only two cores per node, while its successor, Blue Gene/P (BG/P) features four cores per node, and the recently announced Sequoia machine at LLNL will offer close to 100,000 nodes with an estimated 16 cores per node totalling 1.6 million cores [4].

We must understand the application performance impact of this architectural trend of increasing core counts per node. We present initial experiments that study its impact on the Miranda code. We use both a 2D and 3D molecular Dynamics Kelvin-Helmholtz computation and discuss performance results from runs on BG/L, BG/P and a quad-core/quad-socket Linux cluster. Our results show that Miranda can efficiently exploit these current architectures. However, our experiments also indicate that we will require new techniques for future systems with even larger core counts.

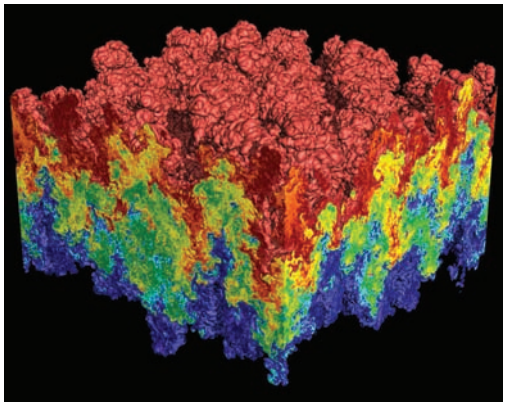


Figure 1: Turbulent state of Rayleigh-Taylor instability. Light fluid (density=1) is blue and heavy fluid (density=3) is red.

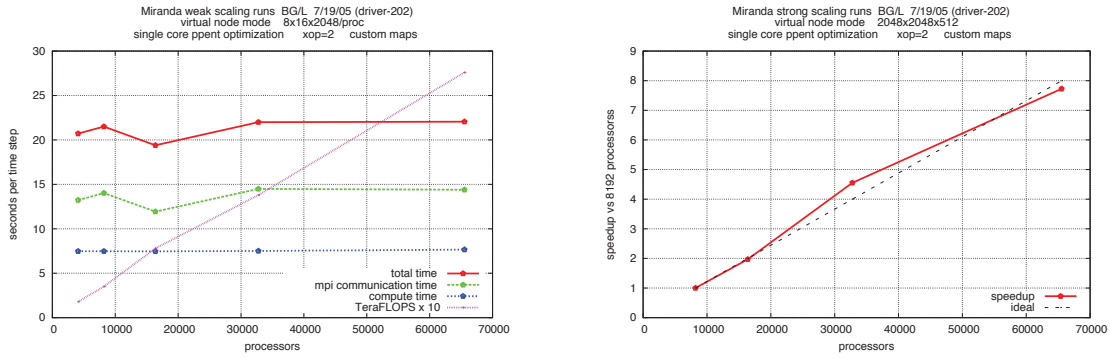


Figure 2: Weak (left) and strong (right) scaling performance of *Miranda* on BG/L. The weak scaling runs had a $16 \times 16 \times 2048$ point grid per node. The strong scaling runs used $2048 \times 2048 \times 512$ total grid points.

2 MIRANDA: DESIGN AND IMPLEMENTATION OVERVIEW

Miranda employs FFTs and band-diagonal matrix solvers for computing spectrally-accurate derivatives, combined with high-order integration methods for time advancement; e.g., fourth-order Runge-Kutta. Fluid properties, i.e., viscosity, diffusivity and thermal conductivity, are computed from kinetic theory. The code contains solvers for both compressible and incompressible flows. Very large simulations, using over 1000 grid points in each direction, are needed to support the large range of length scales necessary to grow R-T and R-M instabilities to full turbulence. Smaller simulations simply cannot capture true rates of growth and mixing due to initial and/or boundary effects.

Miranda's two core tasks are solving FFTs required by the Poisson solve and computing compact implicit derivatives f' from function f. The latter requires solving a linear matrix problem in the form of the equation $Af' = Bf$ with A being a pentadiagonal matrix and B a heptadiagonal matrix. To achieve this, Miranda employs a parallel direct solver method for block distributed pentadiagonal matrices. In this method, local (incomplete) pentadiagonal solutions are performed on each process, boundary data is gathered across the communicator to form a global overlap solution, and then each process uses the overlap solution to complete the global solution. Computing the right-hand side of the equation requires sharing planes of boundary data with nearest neighbors; e.g., for the heptadiagonal matrix B, 3 planes of data must be exchanged at each boundary with paired `MPI_Sendrecv` calls. The pentadiagonal A matrix on the left-hand side of the equation generates 4 planes (2 in each direction) of overlap data from the local solution. The global solution requires all overlap data, so Miranda uses `MPI_Allgather` to collect overlap data among processes. Each process then computes the global overlap solution independently (for load balancing) and completes the exact global solution. The global overlap problem requires the solution of a 4×4 block-tridiagonal linear matrix problem with a dimension equal to the number of processes on the communicator (and hence is not strictly scalable). If the direction is globally periodic, the global overlap solution involves a periodic block-tridiagonal matrix.

Due to the extensive use of FFTs and the connected all-to-all message patterns, Miranda is very communication sensitive. To avoid bottlenecks and to achieve good load balancing, Miranda has been optimized for machines with mesh networks, such as BG/L's torus interconnect. It relies on MPI's Cartesian communicators and maps these communicators directly onto the torus to reduce message time, primarily in *MPI_Alltoallv* calls on subcommunicators. We have replaced all code that operates in a master-slave manner with fully parallel routines, reduced memory overhead, and used an FFTW library tuned for BG/L.

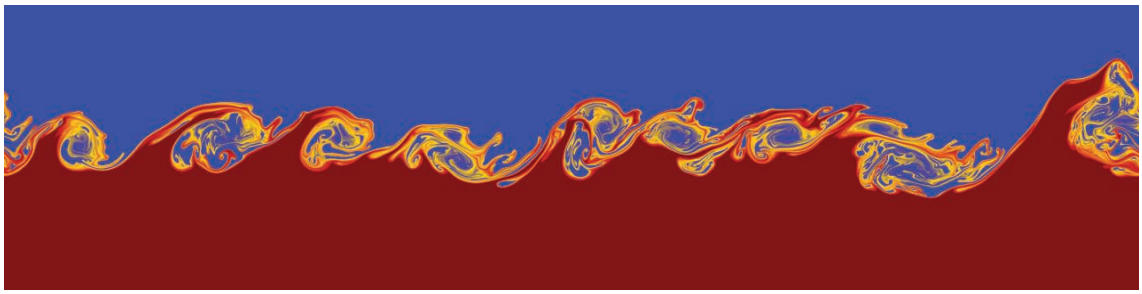


Figure 3: Snapshot of a 2D Molecular Dynamics Kelvin-Helmholtz computation after 2ns (all copper = red/bottom, all aluminiu = blue/top)

	2D-128	3D-4K	3D-16K
Number of CPUs	128	4096	16384
CPU configuration $X \times Y \times Z$	16 x 1 x 8	16 x 32 x 8	32 x 32 x 16
Total working set $X \times Y \times Z$	5120 x 1 x 2560	800 x 1600 x 400	1600 x 1600 x 800
Number of time steps	10	10	10

Table 1: Specification of test problems used in all experiments.

3 EXPERIMENTAL SETUP

We study Miranda's performance on a BG/L system at Lawrence Livermore National Laboratory (LLNL) and on a BG/P system installed at Argonne National Laboratory (ANL). BG/L uses a dual core version of the embedded PowerPC 440 running at 700 MHz as its basic compute node. It then groups those hierarchically into node cards, midplanes, and racks. The total system size used in this study is 40 racks with 1024 compute nodes per rack. BG/L relies on several independent networks, including a 3D torus for point-to-point messaging, a tree network for global collective communication and a separate barrier network.

As a successor to BG/L, BG/P maintains many of its properties; in particular the system provides the same types of networks. The most notable difference is the upgraded compute node, which now features a Quad-core processor built on top of the PowerPC 450 architecture running at 850 MHz.

We also use Hera, a commodity cluster based on AMD Opteron nodes connected with double data rate Infiniband links in order to further study the performance impact of multicore systems. The complete system has 864 nodes, each node with four quad-core chips (16 cores), which results in 13,824 cores in the overall system.

For all of the following experiments we use a Molecular Dynamic Kelvin-Helmholtz (MDKH) computation. Figure 3 shows a sample result of a 2D setup after 2ns of simulated time. We use a 2D problem for small scale runs and two different 3D setups for large scale runs; all scaling results are achieved with weak scaling. Table 1 gives a complete overview of the problems sets used in this work.

All data is measured using the MPI profiling tool mpiP [5], which separately reports time spent inside the application and time spent inside the MPI library.

4 PERFORMANCE RESULTS

Due to memory restrictions and the size of the binary (including debug symbols and the performance analysis tool) we were not able to run in Virtual Node mode on either machine and we therefore restrict ourselves to the results on BG/P, which allowed us to investigate the performance impact of its multicore setup by comparing runs in Dual (running two MPI tasks per node) and SMP (running one MPI task per node) mode. Additionally, we ran one early scaling experiment on BG/P with 16384 tasks. Table 2 shows the results for various processor configurations on BG/P.

Mode	Blue Gene/P SMP	Dual	SMP
Problem	3D-4K	3D-4K	3D-16K
Processors	4096	4096	16384
Application time	4956.06s	4980.47s	6060.79s
MPI time	77.88s	89.60s	332.64
App/MPI ratio	1.57%	1.80 %	5.49%

Table 2: Execution on BG/P for 4096 and 16384 nodes on various processor conifgurations.

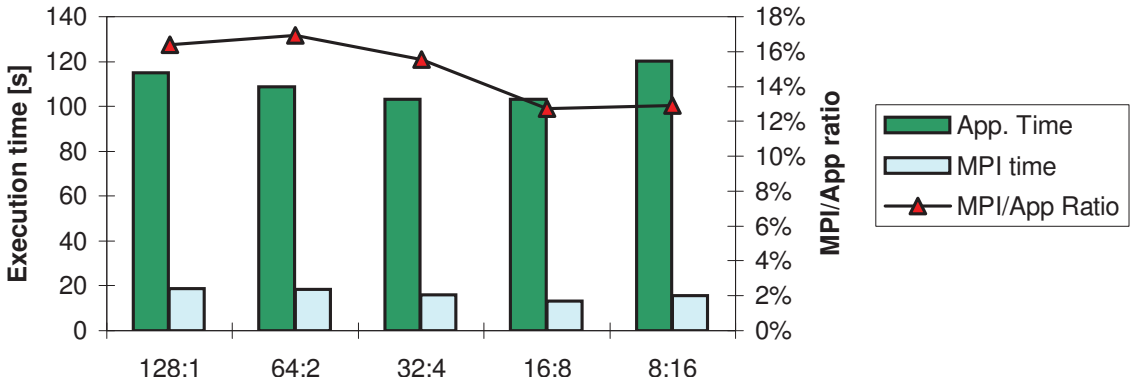


Figure 4: Results of the 2D problem with 128 nodes using different node/multicore configurations (identified by the tuple number of nodes : cores per node) running on Hera.

The results show that Miranda continues to scale well on BG/P systems, however, with slightly reduced efficiency likely caused by the system's smaller communication to computation ratio. Further, these results are very early and the BG/P version has not yet been fully optimized. Looking at the results on 4096 tasks, we can see that Miranda's optimizations successfully ensure a low communication overhead of under 2%. Comparing the two modes, we see an interesting trend: Miranda executes faster when using only one core on 4096 nodes (SMP mode) (idling the remaining three cores per node) than with 2 tasks on each of the 2048 nodes (Dual mode). BG/P's fast interconnection network and Miranda's localization optimizations make up for the possible disadvantage of having more off node traffic. Instead increased overhead caused by on-node communication leads to this counterintuitive effect.

To study this effect further, we ran the smaller and more communication intensive 2D experiment on Hera using 128 MPI tasks split over 128, 64, 32, 16, or 8 nodes utilizing 1, 2, 4, 8, or 16 cores per node respectively. The results are shown in Figure 4. On this machine and for smaller core counts per node (up to 8), performance improves, allowing the efficient utilization of the machine's performance. For 16 cores, however, we see a similar behavior as on BG/P when running on more nodes with fewer cores is beneficial, which can again be attributed to the increased communication overhead being outweighed by the on-chip communication overhead combined with the chip's limited bandwidth to memory and the network.

5 CONCLUSIONS

Efficient execution of scientific applications, such as the Miranda CFD code, faces increasing challenges on today's multi- and many-core architectures. To understand and to evaluate the impact of this architecture trend, we study the performance of Miranda on multicore platforms from the Blue Gene line and a commodity cluster. Initial results show good scaling, but also indicate a possible performance drop for even larger cores per chip configurations. In the future we will extend this study with the goal to optimize the code's multicore behavior and to model and to predict performance on future system architectures.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. It also used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. These resources were made available via the Performance Evaluation and Analysis Consortium End Station, a Department of Energy INCITE project. Neither Contractor, DOE, or the U.S. Government, nor any person acting on their behalf: (a) makes any warranty or representation, express or implied, with respect to the information contained in this document; or (b) assumes any liabilities with respect to the use of, or damages resulting from the use of any information contained in the document. (LLNL-ABS-411404)

REFERENCES

[1] W. Cabot and A. Cook. Reynolds number effects on Rayleigh-Taylor instability with possible implications for type-Ia supernovae. *Nature Physics*, 2:562–568, Aug. 2006.

[2] A. Cook, W. Cabot, M. Welcome, P. Williams, B. Miller, B. de Supinski, and R. Yates. Tera-scalable Algorithmms for Variable-Density Elliptic Hydrodynamics with Spectral Accuracy. In *Proceedings of IEEE/ACM Supercomputing '05*, Nov. 2005.

[3] B. de Supinski, M. Schulz, and et al. Blue Gene/L Applications: Parallelism on a Massive Scale. *International Journal on High Performance Computing Applications (IJHPCA)*, 22(1):33–51, Spring 2008.

[4] Lawrence Livermore National Laboratory. NNSA awards IBM contract to build next generation supercomputer. Press Release, available at https://publicaffairs.llnl.gov/news/news_releases/2009/NR-09-02-01.html, Feb. 2009.

[5] J. Vetter and C. Chambreau. mpiP: Lightweight, Scalable MPI Profiling. http://www.llnl.gov/CASC/mpip/, Apr. 2005.

PARALLEL CFD: PERFORMANCE AND SCALING TOOLS

Performance Engineering: Tools and Techniques for Getting the Most out of your Application.

David C. Cronk*

**The University of Tennessee, Knoxville, TN 37996, USA*
(Tel: 865-974-3735; e-mail: cronk@eecs.utk.edu)

Keywords: Performance, Optimization, Speedup

1. Introduction

With the advent of the multi-core processor, and the use of non-traditional processors (GPGPU, FPGA, Cell) as accelerators on large HPC architectures, these architectures are becoming heterogeneous and much more complex than HPC architectures of the recent past. For example, Roadrunner[1], the worlds fastest supercomputer[2], is comprised of AMD Opteron dual-core processors, each attached with an IBM Cell processor[3]. Since the Cell processor is itself a multi-core processor, the system is a hybrid consisting of multi-cores connected to multi-cores. To complicate matters further yet, the Cell processor is a heterogeneous architecture itself, consisting of a single Power Processing Unit and eight Synergistic Processing Elements. So the Roadrunner system is really a hybrid system consisting of homogeneous multi-cores connected to heterogeneous multi-cores.

It is not uncommon for production quality codes to run at significantly less than 20% of theoretical peak (many dropping below 5%) on conventional HPC architectures. The causes of this poor performance are many and varied. A common cause is what is often referred to as the memory wall. That is, modern architectures cannot move data from memory to the processors quickly enough to keep the processor busy. Other causes include communications overhead, I/O costs, and load imbalance. The increasing complexity of modern architectures will only exacerbate these performance problems. For example, as we move towards higher degrees of multi-core, the memory wall problem only gets worse, making memory management in general, and cache utilization in particular, vital to achieving good performance.

There is a need for tools and techniques to allow users and developers to improve the performance of new and existing codes to better make use of available resources. Since it is common for a single application to run on multiple architectures, we feel it is important for the available tools to work across platforms. It is much more efficient for the developer to learn a tool once and be able to use that same tool across platforms. Still, all the tools in the world will prove useless without an understanding of the techniques needed to find and alleviate performance bottlenecks in large codes. These techniques allow for a systematic approach to performance analysis and optimization.

2. Performance Analysis Techniques

Performance analysis and optimization is a cyclic process. This process starts with characterizing the performance of the whole application. This characterization is then used to search for load imbalance and/or performance bottlenecks. If a load imbalance is detected, the further investigation is carried out in an attempt to find the cause of the load imbalance. Static or dynamic load balancing techniques can be used to try to alleviate the problem. Static techniques may include things as simple as reorganizing the communication pattern, as load imbalance is often caused by processes waiting for data when useful work can be done.

The basic process then involves finding areas of the code that use the most processing time. It makes no sense to try to optimize a routine that only accounts for 1% of the total execution time. Concentrating on areas of the code that use the most time, performance bottlenecks are found, using tools such as those discussed below, and the cause is determined using these or other tools. At this point, changes are made to try to alleviate the bottleneck, and further

performance analysis is performed to determine the affect of the changes, and to further identify performance bottlenecks. This process is repeated with further refinements each iteration.

The performance analysis part of this process involves searching for potential performance bottlenecks. There are many things that can degrade the performance of code. For example, cache performance can have significant performance ramifications. Since the time to access data from main memory can run two orders of magnitude slower than the time to access data from the cache, high cache miss rates can degrade an application's performance to a crawl. Similarly, events such as TLB misses, process stall cycles, branch mis-predictions, etc. can degrade the performance of an application.

Along with measurable events, other factors can affect the performance of an application. Communication performance is often a major cause of performance degradation in parallel application. A process that has useful work to do is wasting time if it is waiting for data from another process. Similarly, poor communication patterns can lead to poor performance by implicit serialization of communication. Similar to communication performance, I/O performance can be critical to good parallel performance.

3. Performance Analysis Tools

There is a vast array of cross platform performance analysis tools available to the developer. Many of these tools are open source and come from the academic community. Others are commercial tools with varying levels of pricing. There are also some tools that offer just a few features, but do them very well, while other tools are monolithic, offering many different features. Often, the choice is personal preference. We present a set of tools that provides varying levels of functionality. One thing each of these tools has in common is a high level of support. While we are comfortable recommending these tools, we recognize there are many equally appealing tools available.

3.1 PAPI

"PAPI aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors. PAPI enables software engineers to see, in near real time, the relation between software performance and processor events."[4] PAPI offers a standard interface to hardware counters. This grants access to valuable profile data including cache misses (at all levels of cache), TLB misses, process stall cycles, floating point instructions, branch instructions, and many more. Additionally, when paired with other tools, this information can be made available at many different levels, including whole program level, routine level, block level, or even instruction level.

3.2 TAU

Tuning and Analysis Utilities (TAU)[5] is a comprehensive package of tools for performance engineering. TAU provides the ability to generate profile data, trace data, or a combination of both. TAU also provides profile analysis tools, both text and GUI based. Further, TAU ships with JumpShot[6] as a tracefile visualization tool. TAU can be integrated with PAPI, allowing for access to hardware counter data, can be used to profile or trace MPI or OpenMP programs (or, even hybrid programs), supports hand or automatic instrumentation, supports multi-language programs, provides program, routine, loop, block, and instruction level profiling, and supports C, C++, Fortran, Python, and Java.

3.3 Scalasca

"SCALASCA is a performance analysis tool being developed with the goal of making the optimization of parallel applications on large-scale systems both more effective and more efficient".[7] While collecting performance data has become easy, analyzing massive amounts of performance data is much more difficult. One of the primary goals of the Scalasca project is to automate this analysis step. Scalasca generates trace files for MPI, OpenMP, or hybrid programs. The analysis step is then performed in parallel, basically replaying the communication of the application, based on the trace file, searching for common performance bottlenecks, such as late senders or late receivers. Once this analysis has been complete, Scalasca provides a visualization tool to view the call-path profile of the execution. This view brings potential performance bottlenecks to the developer’s attention.

4. Example Results

We have worked with a number of applications performing detailed performance analysis studies. This work has led to some significant successes in optimizing production codes. Working within the DoD HPCMP PET program, we have achieved speedups of 3.8X on the molecular dynamics code LAMMPS, 4X on the quantum chemistry package GAMESS, and 4.7X on the computational chemistry package NWChem.

References

1. http://www.lanl.gov/roadrunner/
2. http://www.top500.org/lists/2008/11
3. http://www.research.ibm.com/cell/
4. http://icl.cs.utk.edu/papi/
5. http://www.cs.uoregon.edu/research/tau/home.php
6. http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm
7. http://icl.cs.utk.edu/scalasca/overview/index.html

Multi-Language Instrumentation of CFD Applications using TAU

Sameer Shende*, Allen D. Malony*, Alan Morris*

**ParaTools, Inc., Eugene, OR 97405, USA*
(Tel: 541-913-8797; e-mail: {sameer, malony, amorris}@paratools.com)

Abstract: This paper describes TAU's support for multi-language instrumentation for parallel CFD applications. TAU is an integrated performance evaluation toolkit for application codes written in Python, Fortran, C, and C++, and using MPI for targeting scalable high-performance computing (HPC) platforms. However, applications are more often combining code modules written in different languages, requiring performance measurement support that can span the code ensemble. This paper describes issues of instrumentation across multiple languages, of integration of multi-module measurements, and of performance analysis in multi-language applications. We demonstrate our approach with TAU applied to the CFD application domain.

Keywords: Performance instrumentation, measurement, analysis, TAU, CFD.

1. INTRODUCTION

The ever increasing complexity of parallel systems coupled with advances in software development environments for HPC systems has led to some challenging problems for performance evaluation tools. For computational scientists, programming these systems effectively is a challenging task and often involves using one or more languages, both to apply the best language abstraction for the application module development and to better support integration of application components. Understanding the performance of their parallel applications is equally daunting and the problem does not get easier when multiple languages are applied. To observe and comprehend the performance of parallel applications that run on HPC systems, we need performance evaluation tools that can show performance data for each language and runtime layer with respect to the semantic entities that are meaningful to the scientist. We describe how TAU provides a performance instrumentation layer that can capture semantic entities from multiple compiled and interpreted languages to enable developers to better relate performance to their application behavior.

TAU capabilities are effectively applied to CFD applications where there is a growing trend towards composing large-scale CFD simulations using Python as a driver, such as in the Helios project [1]. The key advantage for using a Python-based substrate in developing large-scale multi-disciplinary codes is greater flexibility in composing the simulations using an interpreted language, as well as runtime resolution of symbols using dynamic shared objects. It is no longer necessary to create a monolithic package that statically binds together pieces of software developed by different groups. Instead, the loose coupling offered by Python helps developers create a plug-and-play environment using shared objects that resolves dependencies at runtime. Not only does this reduce the complexity of application development and compilation, the enhanced runtime configuration and execution flexibility makes it possible to more rapidly experiment with application solutions. This is also true with respect to application performance evaluation and tuning. We show how TAU's multi-language instrumentation and measurement is effective using the Python driver case study for CFD applications.

2. MULTI-LANGUAGE, MULTI-LEVEL INSTRUMENTATION

The TAU Performance System® [2] provides users with robust, portable instrumentation, measurement, and analysis capabilities for observing and evaluating the performance of multi-language CFD codes using Python, C, C++, and Fortran languages. The insertion of hooks for measurement (i.e., *instrumentation*) for multi-language codes that use an interpreted language (e.g., Python or Java) with traditional compiled languages requires special care. For instance, computational kernels are typically loaded in Python using dynamic shared objects, or shared libraries that provide an interface with Python. Parallelism is achieved by distributing the computation over multiple processors using a Python package that interfaces with MPI, such as pyMPI [3] or mpi4py [4]. *Events* (i.e., actions that take

place in the program) must be exposed (through instrumentation) to the performance measurement facility in a uniform manner independent of the language where the events are generated. These event measurements must flow into a common performance data repository maintained within each executing context. Events triggered by the Python interpreter, such as routine entry or exit, must seamlessly blend in with events generated by computational kernels coded in Fortran and C. These, in turn, must interface with MPI or other runtime layer events so a common calling stack is maintained within the performance tool.

The mechanism for code instrumentation may differ significantly for each of these layers. For instance, we measure the performance of MPI libraries by intercepting the calls made by the application to the MPI library. A tool such as TAU provides a wrapper interposition library that defines an MPI interface using weak bindings and a name-shifted interface [2]. In contrast, the Python interpreter provides hooks for instrumentation that TAU uses to capture routine transition events while interpreting the code at runtime, and the source code of the CFD application is typically instrumented using compiler-based instrumentation or a source-to-source translator. Instrumentation methods involve setting some environment variables to identify the measurement options chosen, and substituting the compiler used by the build system with a TAU compiler script (*tau_f90.sh*, *tau_cxx.sh*, and *tau_cc.sh* for Fortran, C++, and C respectively).

While creating a shared object of the computational kernel or while linking in the pyMPI executable, TAU's shared object must be linked in. When events from Python, pyMPI, C, C++, or Fortran invoke TAU's measurement calls, TAU's dynamic shared object provides the necessary symbol resolution of the measurement substrate and a single performance data repository is used. While source-based instrumentation embeds the name of the routine in the instrumentation call as a parameter, compiler-based instrumentation requires dynamic mapping of the routine identifier (typically the address) to its name using the GNU BFD package under Linux. Python based codes load and unload packages that are implemented using shared objects. When these packages use compiler-based instrumentation in TAU, the mapping of a symbol address to its name must be maintained by TAU at runtime. TAU updates these address, name mappings as libraries are loaded and unloaded and de-mangles routine names as necessary for C++ codes. The result is shown in Figure 1 that shows a profile from TAU showing events from Fortran, C, and C++ languages and runtime systems such as MPI and pyMPI.

3. CONCLUSIONS

The complexity of CFD applications and high-end computers system on which they run demand a high level of sophistication in parallel performance tools, both their robust capabilities and the methodological practices for their effective use. The paper highlights the support for instrumentation of multi-language CFD applications using TAU.

4. ACKNOWLEDGEMENTS

This publication was made possible through support provided by DoD HPCMP PET activities through Mississippi State University under the terms of Agreement No. GSO4TO1BFC0060**.** The opinions expressed herein are those of the author(s) and do not necessarily reflect the views of the DoD or Mississippi State University. All information contained herein was obtained from open sources.

REFERENCES

1. Wissink, A., and Shende, S. (2008) Performance Evaluation of the Multi-Language Helios Rotorcraft Simulation Software, DoD HPCMP UGC 2008 conference, Seattle.
2. Shende, S., and Malony, A. D. (2006) The TAU Parallel Performance System, *International Journal of High Performance Computing Applications*, SAGE Publications, 20(2), 287-331. http://tau.uoregon.edu
3. http://pympi.sourceforge.net/
4. http://code.google.com/p/mpi4py/

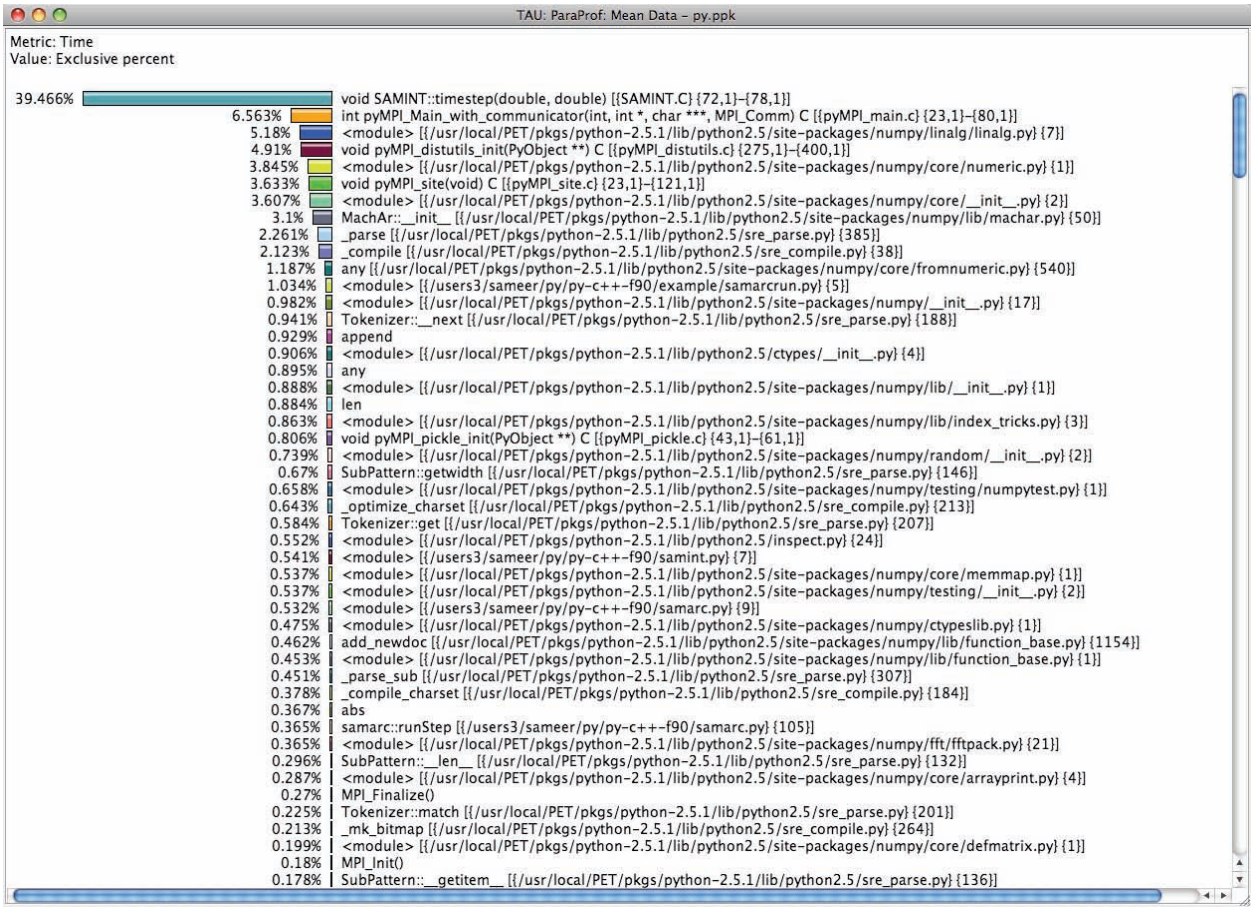


Fig. 1: *Multi-level instrumentation using TAU exposes events from Python, C++, C, Fortran and MPI.*

Parallel Performance Evaluation of Helios

Andrew M. Wissink*, Sameer Shende**
*Scaled Numerical Physics LLC, Ames Research Center, Moffett Field, CA 94035, USA
(Tel: 650-604-5385; e-mail: andrew.m.wissink@us.army.mil)
**Paratools, Inc., Eugene, OR 97401 USA
(e-mail: sameer@paratools.com)

Abstract: This paper describes parallel performance considerations of Helios, a software package used for high-fidelity aero/structural dynamics analysis of rotary-wing vehicles. Helios consists of a suite of CFD and CSD software modules integrated through a high-level Python-based software integration framework. This loosely-connected approach has the advantage that each module can be developed separately from one another, but as with any parallel code, a single poor-performing module can hinder the performance and scalability of the suite as a whole. We utilize the the TAU performance system® to assess the performance of different modules, demonstrating ways to measure both single processor execution and and parallel scaling performance from the Python level, without adding trace calls or recompiling the underlying executables. Both the execution rate and parallel performance of routines within the code modules can be discerned.

Keywords: Computational Fluid Dynamics, Python, Parallel Computing

1. INTRODUCTION

Helios is the software product for high-fidelity multi-disciplinary analysis of rotorcraft being developed by the HPC Institute for Advanced Rotorcraft Modeling and Simulation (HI-ARMS) [1,2]. Instead of the traditional model where different physics are coupled together within a single code, Helios links together separate physics modules - computational fluid dynamics (CFD), computational structural dynamics (CSD), moving body dynamics, and various interface routines - through a high-level Python-based integration framework. The integrated set of modules runs on large-scale parallel HPC systems.

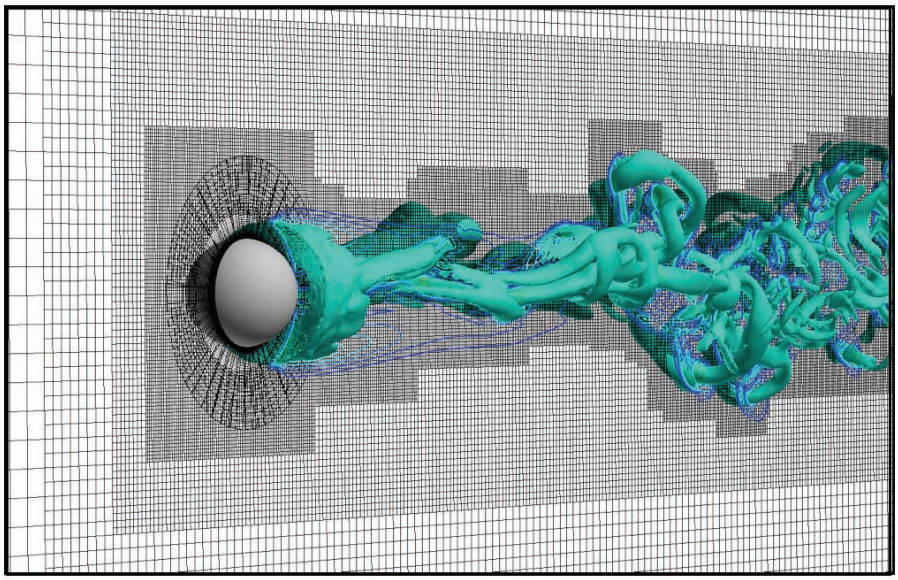


Fig. 1: *Near-body/Off-body overset meshing paradigm used by Helios.*

Rotary-wing CFD analysis requires accurate resolution of viscous boundary layer flow near the blades, but also resolution of the wake structure since tip vortices interacting with rotor blades have a profound impact on vibration, noise, and handling qualities. Unfortunately, no single CFD paradigm is optimal for these different domains. Unstructured solvers are useful for resolving complex geometries and capturing the viscous boundary layer, but they generally do a poor job with the wake because they are computationally expensive and limited to 2^{nd}-Order accuracy, which causes the wake to dissipate quickly. Cartesian solvers do a poor job in the viscous boundary layer but are good at capturing the wake because they can be readily extended to high-order, are very computationally efficient (up to 10X faster than unstructured in our tests), and are amenable to automated adaptive mesh refinement (AMR) strategies. Hence, the approach taken in Helios is to apply an unstructured solver near the body surface and adaptive high-order solver throughout the rest of the domain (Fig. 1). This leads to a highly-accurate prediction of both the loads and the wake in a fast and efficient manner.

Two CFD modules are used in Helios. Near the body surface, the Reynolds-averaged Navier-Stokes solver NSU3D [3] is applied. The farfield wake is resolved using a high-order block structured adaptive Cartesian Euler solver SAMARC. SAMARC couples the SAMRAI [4] package from Lawrence Livermore National Laboratory to manage adaptive grid generation, parallel load balancing, and inter-processor, with the high-order ARC3DC [5] Cartesian block solver from NASA Ames. The PUNDIT [6] domain connectivity software manages chimera-style interpolation between the different grid systems. Structural loads and six-degree-of-freedom information are supplied by the RCAS software [7]. Further details on the implementation and validation of Helios are in references [1,2]. Although this multi-code strategy offers the advantages of "plug-and-play" for different simulation scenarios, it makes parallel implementation and load balancing much more complex than with a traditional monolithic simulation code.

2. PERFORMANCE EVALUATION

The key advantage for using a Python-based substrate in developing large-scale multi-disciplinary codes is greater flexibility in composing the simulations using an interpreted language, as well as runtime resolution of symbols using dynamic shared objects. It is no longer necessary to create a monolithic package that statically binds together pieces of software developed by different groups. Instead, the loose coupling offered by Python helps developers create a plug-and-play environment using shared objects that resolves dependencies at runtime. To better understand the performance characteristics of such a framework, both profiling and tracing are relevant. While profiling shows summary statistics, tracing can reveal the temporal variation in application performance.

TAU [8] provides users with robust instrumentation, measurement, and analysis capabilities for observing and evaluating the performance of HPC applications. Of particularly importance for a code structure like Helios, TAU supports automatic instrumentation at multiple levels; the Python runtime level, the MPI library level and in the different languages at the source code level. Source code is pre-processed by TAU and calls to the API (for starting and stopping timers) are automatically added at the entry and exit of each routine, and if desired outer loops, during the compilation process. The advantage of Python based instrumentation is that it requires no intervention on the part of the developer. The same executables may be used as in the un-instrumented case, but run through an alternative instrumented version of pyMPI. The advantage of the source-code instrumentation approach is that it gives more detailed information, but a special wrapped compiler script must be used at compile time.

We used TAU to automatically instrument Helios at the Python, pyMPI, MPI, and source-code level. To improve its coverage, the instrumentation at the different code levels must be compatible, and performance data streams generated at each level must merge to form a unified performance data state within the address space of the executing application.

2.1 Python-level Instrumentation

A multi-processor Helios job is executed using a Python script run under pyMPI [9]. TAU has built a version of pyMPI which automatically tracks all calls into and out of Python and MPI routines. Thus, generating a performance trace is simply a matter of executing the script with the TAU-enabled version of pyMPI. Figure 2 shows the performance traces for two CFD applications generated using this approach. The first computes hover conditions about a quarter-scale V-22 rotor (experimental model referred to as TRAM) on 16 processors, using NSU3D for the near-body solver and SAMARC for the off-body. The second computes flow over a NACA 0015 on

32 processors and applied TURNS in the near-body and SAMARC in the off-body. In both cases, grid adaptivity was applied in SAMARC to resolve tip vortices. The profile plots are generated by TAU's *paraprof* tool.

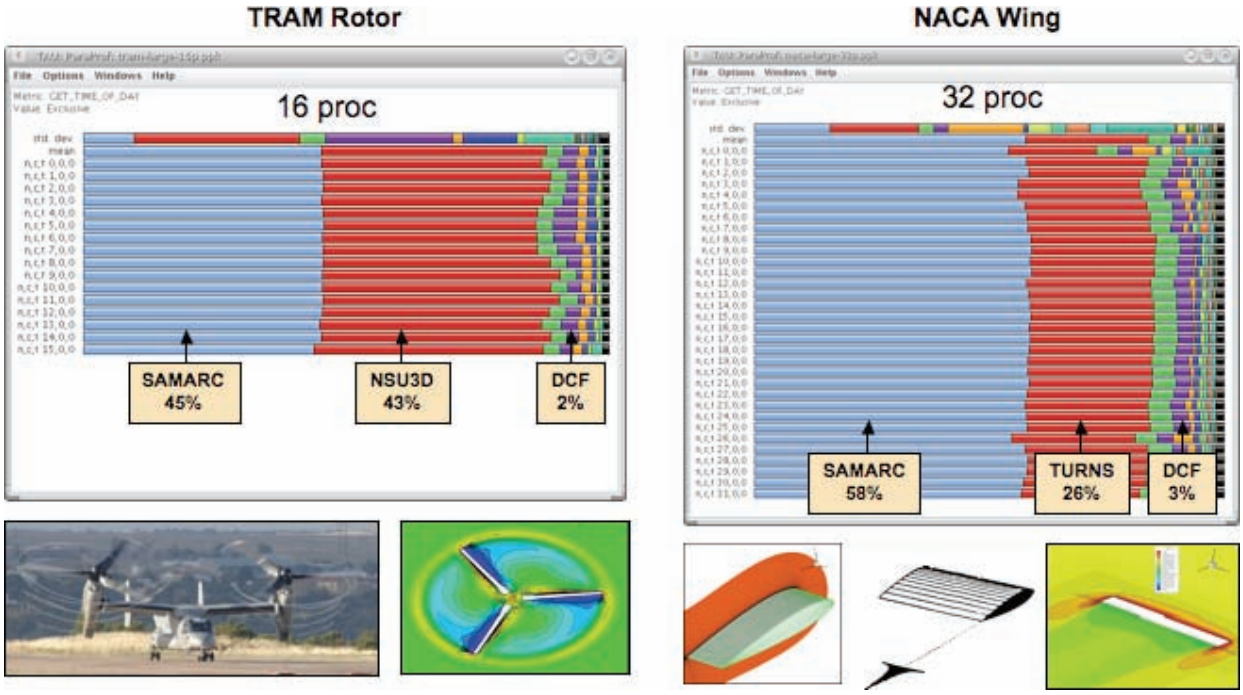


Fig. 2: *Trace analysis performed using TAU's paraprof tools from Python-level instrumentation of Helios modules. CFD analysis was performed on ARL's mjm system of a TRAM (quarter-scale V22) rotor in hover on 16 processors, and flow over a NACA 0015 wing on 32 processors.*

A lot of useful information can be derived from this simple Python-level instrumentation. It is clear from the profiles what percentage of time is spent in each CFD module, and also the degree of imbalance in the computation. Considering this level of instrumentation requires no change to the application's executables, this is a very effective way to report the performance of a Helios application with essentially no intervention to the codes themselves.

2.2 Compile-time Instrumentation

The Python-level instrumentation can provide some useful insight but it is limited to high-level calls. In the cases shown above, between 85% to 90% of the time is spent in the solvers NSU3D, TURNS, and SAMARC. It is desirable to dive deeper into these routines in order to investigate the performance in more detail, and for this we use compile-time instrumentation. A TAU-wrapped version of the compiler automatically wraps each function in C/C++ code, and each subroutine in F90 code. The developer does not need to manually instrument the code but replaces the C/C++/Fortran compiler normally used to compile the application with a tau-wrapped version that automatically applies the instrumentation. This enables subroutine-level instrumentation of all modules compiled with this approach, facilitating a more detailed look at the routines that take the most time inside the module.

TAU can be configured to record megaflop rates using wallclock time and hardware counters from the PAPI libraries [10]. The hardware counters include metrics such as floating point instructions, data cache misses, etc. The *paraprof* tool allows different recorded metrics to be compared to one another, which reveals other useful execution information besides just wallclock time. TAU also enables organization of the timers and hardware counters into "groups". This is useful for an application like Helios which uses different modules, in order to isolate statistics for a particular module. Figure 3 shows an example using the PAPI hardware counters and groups functionality in order to analyze performance. We tracked the number of floating point instructions, obtained from PAPI, along with the measured wallclock time to obtain statistics on the execution rate of each routine in the module. The four fastest routines in SAMARC run between 850-1500 Megaflops, the fifth to eighth fastest routines run 200-550 Megaflops, and the remainder run at 100 Megaflops or less. The breakdown in NSU3D is a little more

evenly spread, with the ten fastest routines running 465-1200 Megaflops, with a relatively even reduction after that. This information is very useful for identifying sources of single processor inefficiencies.

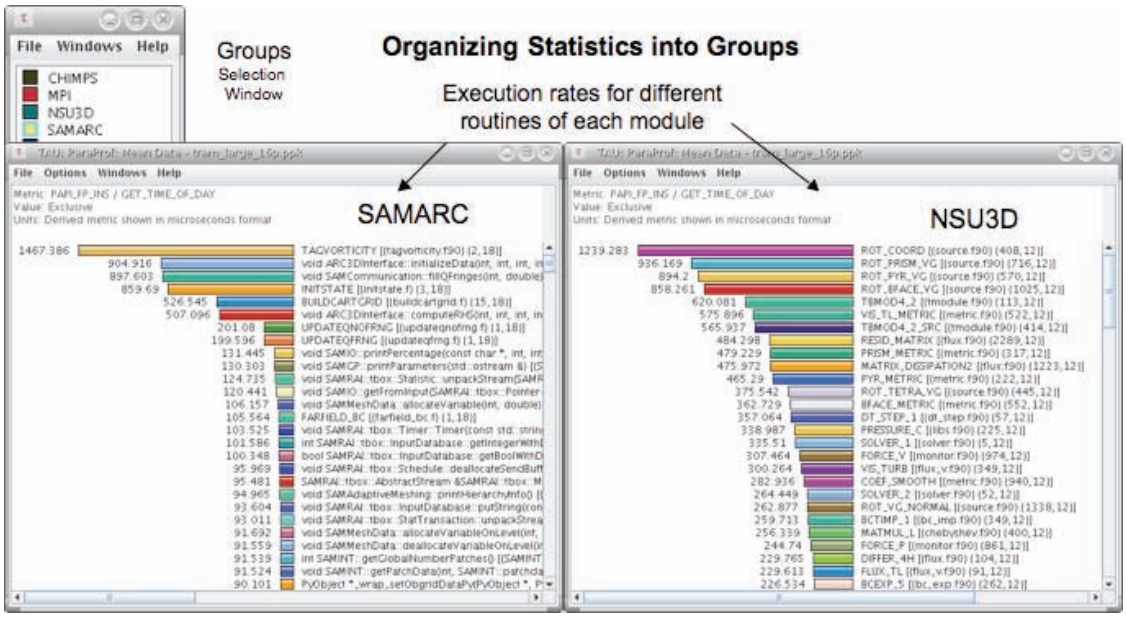


Fig. 3: *The mean execution rates (Millions of floating point instructions per second – Megaflops) of the different routines in SAMARC and NSU3D.*

2.3 Scalability Analysis

The aforementioned techniques can isolate where computational time is spent and the relative computational efficiency of dominant routines within the code. Developers are also interested in the scalability of particular routines, and of course the code as a whole. For this analysis, we use TAU's tool *perfexplorer* [11].

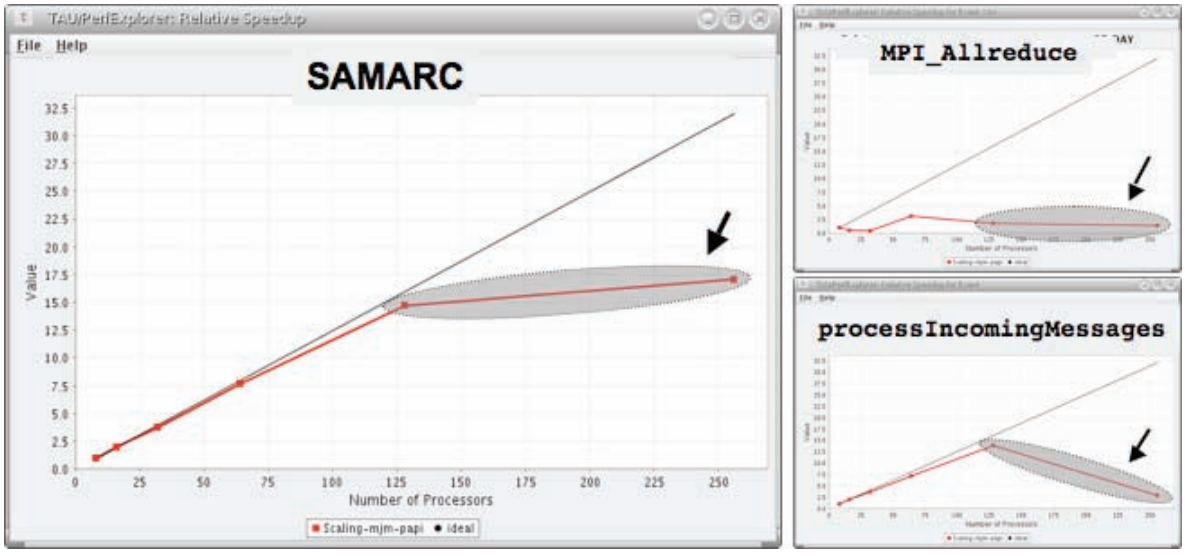


Fig. 4: *Perfexplorer analysis of the SAMARC multi-processor data. Scaling drops off in the transition from 128 to 256 processors. Analysis of scaling in individual routines reveals the cause to be poor scaling in* `MPI_Allreduce` *and* `processIncomingMessages`.

The *paraprof* tool can load performance data from different cases. It supports various database backends such as Derby, a file system based database, postgreSQL, mySQL, Oracle, and DB2. The *perfexplorer* tool may then be used to analyze this data stored for multiple runs to assess performance characteristics across a range of cases. For example, Fig. 4 shows the scaling performance of SAMARC up to 256 processors. There is a clear dropoff after 128 processors. Analysis of the routines revealed that two, `SAMRAI::processIncomingMessages` and `MPI_Allreduce`, were showing particularly poor scaling which was reducing the overall performance of the code. Isolating performance issues in this way makes it much easier for the developer to isolate performance issues early in the development.

4. CONCLUSIONS

We demonstrated the use of TAU tools to perform multi-level performance analysis of the Helios code. High-level instrumentation is applied at the Python layer and is automatic, requiring no modification of the underlying executables. Python-level instrumentation provides useful data on time spent in the modules, identifying load imbalances and communication costs, generally gives a high-level view of the parallel execution. In order to isolate particular routines within the module that are causing inefficiencies, lower-level compile-time instrumentation can be applied. Although this requires recompilation, it provides a wealth of useful information on execution rates and scaling qualities of individual routines within the module, and is therefore useful for isolating performance issues within a large code. We demonstrated how these tools were used to identify sources of inefficiency in Helios.

ACKNOWLEDGEMENTS

Material presented in this paper is a product of the CREATE-AV Element of the Computational Research and Engineering for Acquisition Tools and Environments (CREATE) Program sponsored by the U.S. Department of Defense HPC Modernization program office. Support with TAU was provided by PET activities through Mississippi State University under the terms of Agreement No. #GS04T01BFC0060**.** Jay Sitaraman, Venke Sankaran, Buvana Jayaraman, Anubhav Datta, Dimitri Mavriplis, Mark Potsdam, Hossein Saberi, and Roger Strawn all contributed to this work through their development efforts on Helios.

REFERENCES

1. A.M. Wissink, J. Sitaraman, V. Sankaran, T. Pulliam, and D. Mavriplis, “A Multi-Code Python-Based Infrastructure for Overset CFD with Adaptive Cartesian Grids,” AIAA-2008-0927, 46th AIAA Aerosciences Conference Reno NV, Jan 2008.
2. J. Sitaraman, A. Katz, B. Jayaraman, A. Wissink, and V. Sankaran, “Evaluation of a Multi-Solver Paradigm for CFD using Unstructured and Structured Adaptive Cartesian Grids,” AIAA-2008-0660, 46th AIAA Aerosciences Conference, Reno NV, Jan 2008.
3. D. J. Mavriplis, “Results from the Third Drag Prediction Workshop using the NSU3D Unstructured Mesh Solver,” AIAA-2007-0256, 45th AIAA Aerosciences Conference, Reno NV, Jan 2007.
4. R.D. Hornung, A.M. Wissink, and S. R. Kohn, “Managing Complex Data and Geometry in Parallel Structured AMR Applications,” *Engineering with Computers*, Vol. 22, No. 3-4, Dec. 2006, pp. 181-195. Also see http://www.llnl.gov/casc/samrai.
5. T. H. Pulliam, “Euler and Thin-Layer Navier-Stokes Codes: ARC2D and ARC3D,” Computational Fluid Dynamics Users Workshop, University of Tennessee Space Institute, Tullahoma TN, March 1984.
6. J. Sitaraman, M. Floros, A.M. Wissink, and M. Potsdam, “Parallel Unsteady Overset Mesh Methodology for a Multi-Solver Paradigm with Adaptive Cartesian Grids,” AIAA-2008-7117, 16th AIAA Applied Aerodynamics Conference, Honolulu HI, Jan 2008.
7. *RCAS Theory Manual, Version 2.0*, United States Army Aviation and Missile COMmand/ AeroFlightDynamics Directorate (USAAMCOM/AFDD) Technical Report 02-A-005, US Army Aviation and Missile Command, Moffett Field, CA, June 2002.
8. S. Shende and A. D. Malony, “The TAU Parallel Performance System,” Int’l Journal of High Performance Computing Applications, SAGE Publishers, 20(2): pp. 287- 331, Summer 2006.
9. SourceForge, “pyMPI: Putting the py in MPI”, http://pympi.sourceforge.net, 2008.
10. S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, “A Portable Programming Interface for Performance Evaluation on Modern Processors,” Int’l Journal of High Performance Computing Applications, 14(3), pp. 189-204, 2000. http://icl.cs.utk.edu/papi/
11. K. Huck and A. Malony, “PerfExplorer: A Performance Data Mining Framework for Large-Scale Parallel Computing,” Proc. ACM/IEEE Conference on Supercomputing (SC’05), 2005.

Analyzing the Performance of Scientific Applications with Open|SpeedShop

Martin Schulz*, Jim Galarowicz, Don Maghrak**, William Hachfeld**, David Montoya***, Scott Cranford*****

* *Lawrence Livermore National Laboratory, Livermore, CA, USA*
** *Krell Institute, Ames, IA, USA*
*** *Los Alamos National Laboratory, Los Alamos, NM, USA*
**** *Sandia National Laboratories, Livermore, CA, USA*

(e-mail: schulzm@llnl.gov, {jeg, dpm, wdh}@krellinst.org, dmont@lanl.gov, scranfo@sandia.gov)

http://www.openspeedshop.org/

Abstract: Open|SpeedShop (O|SS) is a comprehensive, open source parallel performance analysis tool set that provides most common performance analysis step in a single environment. It can help application programmers to analyze, understand, and optimize the performance of their codes. O|SS provides a wide range of performance experiments and covers both sampling and tracing techniques and works on sequential, MPI, and multithreaded codes without requiring source code modifications. The latter allows users an easy integration of O|SS into their application's workflow and runtime environment, which ensures a low learning curve and is essential for achieving a wide spread adoption of any performance tool sets.
Keywords: Performance Analysis, Profiling, Tracing, Optimization

1 INTRODUCTION

Performance analysis and optimization is a critical step in the development process of any scientific application. Efficient and easy-to-use tool support is essential to allow users to complete this task, yet current tools are often targeted for the performance analysis expert, and therefore cumbersome to use or require changes to the compilation or execution process. This makes these tools unattractive for application developers who often have limited time allocated for performance optimizations.

To overcome these problems, we have designed Open|SpeedShop (O|SS), an open source multi platform Linux performance tool. It directly targets application developers and computer scientists as the main users and provides easy access to an application's performance profile, while not precluding more sophisticated and detailed analysis found in other tools. O|SS currently runs on Linux clusters with Intel or AMD processors with a wide variety of Linux distributions. O|SS's performance analysis functionality includes a set of specific "experiments" that allow the user to easily gather a variety of different performance statistics about an application. This includes Program Counter Sampling, a light weight way to get an overview of application performance bottlenecks; Call Stack Sampling Analysis, a technique to find hot call paths; Hardware Performance Counters, providing access to low level information such as cache or TLB misses; MPI Profiling and Tracing, enabling users to detect MPI communication bottlenecks; I/O Profiling and Tracing to study an application's I/O characteristics; and Floating Point Exception Analysis to detect floating point exceptions which can slow down applications.

O|SS gathers its performance data from unmodified binaries and includes both offline and online instrumentation and data collection facilities. Once collected, it displays the data through a set of detailed reports that allow the user to easily relate the performance information back to their application source code. This information is accessible through a comprehensive GUI, from a command line interface, as well as from within Python scripts. Additionally, the tool set includes a series of analysis techniques, including outlier detection, load balance analysis, and cross experiment comparisons. In summary, O|SS's functionality provides a comprehensive set of techniques that greatly aid in the analysis and understanding of parallel application performance.

2 THE CONCEPT OF EXPERIMENTS

The central concept in O|SS's workflow is an *Experiment*. It defines what is being measured and/or analyzed. Users select their experiment at the beginning of any performance analysis run depending on what kind of performance bottleneck they would like to investigate. With that, they implicitly choose how the data will be extracted from the application, in which form it will be stored, and what kind of views are available for its analysis.

By default, the tool set offers three sampling and three tracing experiments covering most common needs for performance analysis. These are

- **PC Sampling**, which provides a statistical overview of where a code spends its time and hence offers a good first overview of a code's behavior;
- **User Time**, which adds stack trace information to the gathered samples and thereby adds context on how a code reached observed bottlenecks;
- **Hardware Counter**, which enables users to measure information offered by the CPU's performance counters like cache or TLB misses;
- **I/O tracing**, which gathers a detailed trace of all POSIX I/O operations found in a code and the time they took;
- **MPI tracing**, which provides the same type of information, but for MPI calls executed during an applications execution; and
- **FPE tracing**, which tracks floating point exceptions triggered by the FPU.

Except for the MPI tracing experiment, which naturally requires parallel execution in an MPI environment, all other experiment can be used in either sequential or parallel (MPI or multithreaded) environment. In the latter case, O|SS keeps track of which performance information was generated by which process or thread and then offers the data either in an aggregated (global) view or on a per task/thread group basis.

In addition to the existing default experiments, O|SS allows the addition of new experiments through a plugin infrastructure. By specifying separate plugins for data collection, data preparation, and data presentation programmers can add new tool functionality without having to reimplement the (often complex) surrounding infrastructure. This includes components like instrumentation, data conversion, transport, and storage, and user interfaces. This capability allows O|SS to be extensible and also meet specialized needs, e.g., for particular hardware platforms with special needs or performance monitoring capabilities.

3 USER INTERFACES, ARCHITECTURE, AND WORKFLOW

O|SS offers three different user interface, which are shown in Figure 1: a graphical user interface with source code browser and simple graphing and visualization panels; a specialized scripting language similar to ones used by debuggers like *gdb*; and a Python module that allows the invocation of O|SS from within Python scripts. All three user interfaces are equivalent in their functionality and capabilities and can interact. Data gathered using one interface can be read and analyzed by any other. Further, activities in one interface can be tracked and replayed in any other.

This full interoperability is made possible through O|SS's architecture, which layers all user interfaces on top of a single *Command Line Interface* and performance analysis framework. Underneath this framework, O|SS relies on several open source infrastructures, like SQLite, QT, and Python, as well as two different data collection architectures: one that collects data offline into temporary files and postprocesses them for analysis in O|SS; and one that collects all data online through a scalable communication network. The former is generally more lightweight and easier to handle, while the latter offers the ability to monitor an applications progress and to attach to applications at runtime. Which collection mechanism to use, hence depends on the targeted performance analysis scenario.

Independent of this choice, however, users generally follow the workflow depicted in Figure 2: they first select their target application, the experiment, as well as the data collection mechanism. O|SS then loads the application or attaches to it, instruments it with the necessary performance probes, and runs the application. Depending on the collection

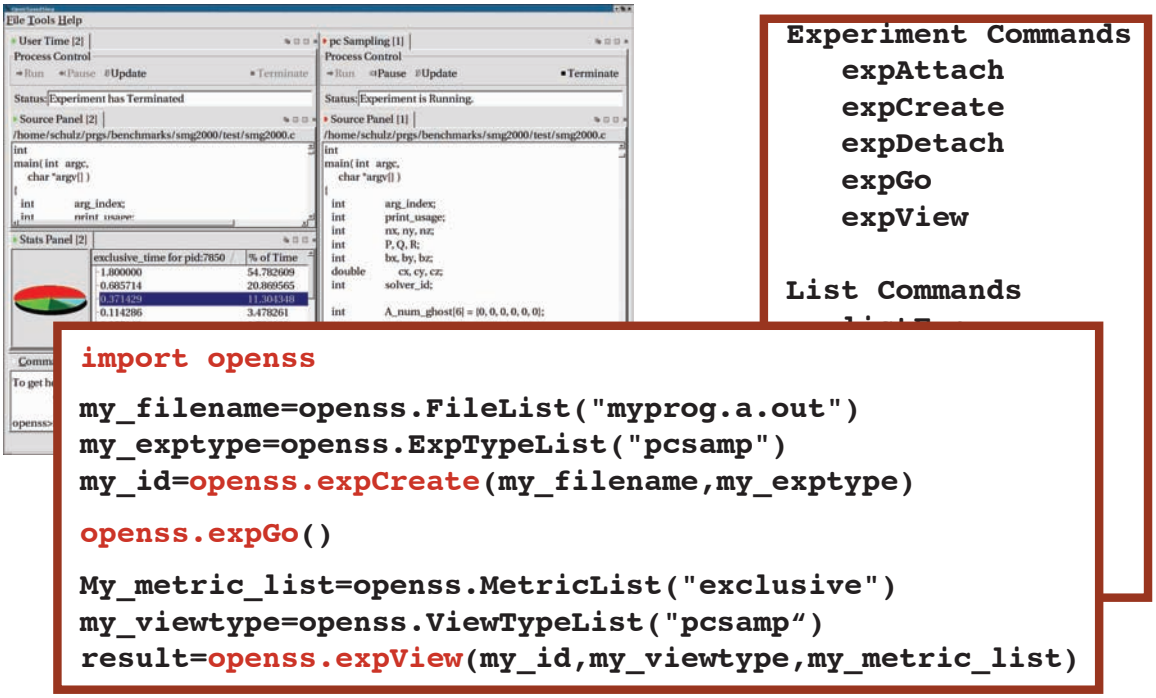


Figure 1: Open|SpeedShop provides three interoperable user interfaces.

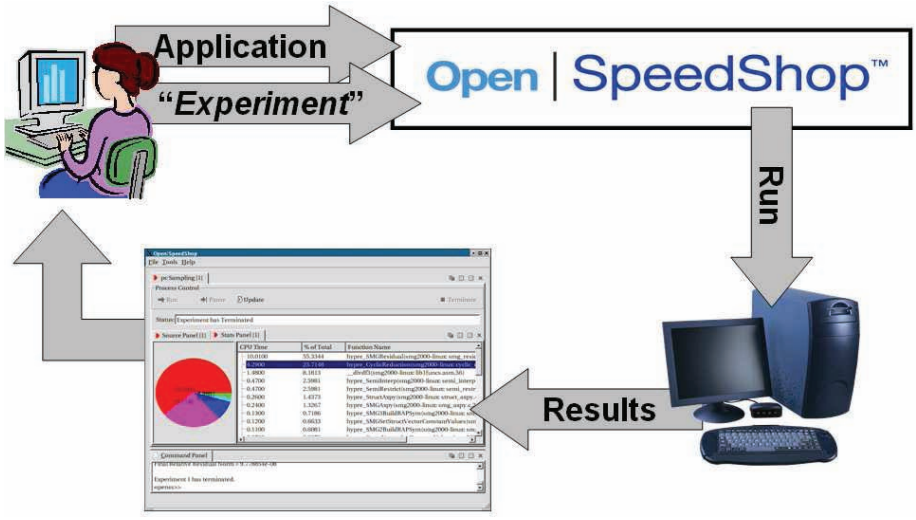


Figure 2: Typical Workflow in O|SS.

tasks	nodes	pcsamp	hwc	io	iot
64	8	0.3%	0.2%	0.0%	1.5%
128	16	0.3%	0.3%	0.0%	0.0%
256	32	0.4%	0.4%	0.0%	0.3%
512	64	0.5%	0.6%	0.1%	2.4%
1024	128	0.4%	0.4%	-	-

Table 1: Overheads when executing O|SS on SMG2000 using a variety of experiments.

mechanism, the performance data is either stored in temporary files and converted after the run's completion, or is directly stored in an SQL database. Once the application terminates or the user polls for intermediate data in the online mode, O|SS reads the collected data from the displays it to the user or performs additional analysis on it. The latter includes cross experiment comparisons, multi rank analysis, or clustering techniques. In any case, the user can use this to gather information about his/her application and then apply this information to either initiate a next set of experiments or to modify the code.

4 SCALABILITY AND OVERHEAD

Performance tools need to be designed to exhibit minimal overhead, in order to not perturb the execution of the application to a point that it significantly alters the application's characteristics. Otherwise, the results obtained from the tool do not reflect the real performance of the application and hence can lead to wrong conclusions.

To study the overhead caused by O|SS we apply it to a set of runs with SMG2000, a Semicoarsening Multigrid Solver based on the hypre library [1], taken from the ASCI Purple benchmark suite [2]. We use an input size $N = 90^3$ for all experiments and we execute them on Hyperion [3], a 1152 node Linux cluster. Each node is equipped with dual socket Intel Xeon Quad-core processors and 8GB of main memory, interconnected with Quad data rate Infiniband.

We investigate the performance of both sampling and tracing experiments and Figure 1 shows the results of four different experiments comparing the execution of the code with O|SS to a baseline without instrumentation. Sampling experiments are expected to produce a uniform overhead across the whole application. For O|SS we see an only negligible overhead of 0.3%-0.6% for the PCsampling and the hardware counter (hwc) experiments. Further, our tool framework scales well showing only a slight increase in overhead when scaling the application run from 64 to 1024 tasks.

The performance of tracing experiments, on the other hand, depends heavily on the activity within the application that is being traced. Typically, tracing experiments capture large amounts of data and are expected to create significant overhead. The results in Figure 1 show the performance of the I/O tracer with and without stack tracer enabled. SMG2000 does only minimal I/O and hence the numbers reflect the overhead required by O|SS to setup the experiment and prepare the trace information, but not the actual recording of trace data. Again, we see a very low overhead and a good scalability in all cases, but using stacktraces can in some cases created a slightly higher overhead of 2.5%.

5 CONCLUSIONS

O|SS is an open-source performance analysis tool set for Linux clusters. It provides a range of experiments for sampling and tracing of performance data for sequential, MPI parallel, and multithreaded codes. Users can access the tool through three separate, yet fully interoperable interfaces. O|SS can be applied to existing binaries without requiring any changes to source code or recompilation and can hence be applied easily to any application. Using a case study of a multigrid solver, we have shown that the overheads caused by the tool are negligible and that the framework supporting O|SS is scalable. Future works includes porting O|SS to new platforms, including machines like BG/L, BG/P, and Cray's XT line, as well as further scalability enhancements. This will develop O|SS into true cross-platform performance analysis tool set that can be applied by end users at any step of their code development efforts, be it on commodity clusters, small test configurations, or production environments on high-end systems.

ACKNOWLEDGMENTS

Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. (LLNL-ABS-412292)

REFERENCES

[1] R. Falgout and U. Yang. hypre: a Library of High Performance Preconditioners. In *Proceedings of the International Conference on Computational Science (ICCS), Part III, LNCS vol. 2331*, pages 632–641, Apr. 2002.

[2] Lawrence Livermore National Laboratory. The ASCI purple benchmark codes. http://www.llnl.gov/asci/purple/benchmarks/limited/code_list.html, Oct. 2002.

[3] Lawrence Livermore National Laboratory. Lawrence Livermore teams with computing industry leaders to develop an advanced technology cluster testbed. Press Release, available at https://publicaffairs.llnl.gov/news/news_releases/2008/NR-08-11-04.html, Nov. 2008.

ENABLING COMPUTATIONALLY BASED ACQUISITION ENGINEERING OF AERONAUTICAL DEFENSE SYSTEMS

Computationally Based Engineering for Air Vehicle Acquisition: The CREATE-AV Project

Robert L. Meakin *

** DoD High Performance Computing Modernization Program, Lorton, VA 22079, USA (Robert.Meakin@hpcmo.hpc.mil)*

Abstract: The CREATE-AV (Air Vehicles) Project is one of three primary elements of the CREATE Program, established in FY2008 by the Department of Defense (DoD) Director of Defense Research and Engineering to improve engineering processes for acquisition of major new military weapon systems. The CREATE-AV Project will develop and deploy a set of Computationally Based Engineering (CBE) software tools for the air vehicles acquisition engineering workforce.

Keywords: Software Engineering, Agile, HPC, Verification and Validation

1. INTRODUCTION

The CREATE-AV software is being developed in response to regularly updated and prioritized requirements to address key capability gaps in acquisition processes of this community. The software aims to (a) increase the capacity of acquisition program engineers, (b) reduce workloads through streamlined and more efficient acquisition engineering workflows, and (c) minimize the need for rework due to early detection of air vehicle design faults and performance anomalies.

The DoD Air Vehicles acquisition capability gaps identified to date by the CREATE-AV Planning Team can be categorized into three broad classes: Conceptual Design, Design Verification, and Design Environment. Each of these represent broad topic areas and demand resources beyond that available through CREATE to fully address. Still, the focused strategic goals of the CREATE-AV project are expected to provide significant, measurable, and necessary benefits to the acquisition engineering workforce and stakeholder organizations. Each of the specific CBE software elements of the CREATE-AV Project addresses these gap classes in an important way.

2. CONCEPTUAL DESIGN CLASS

The *Conceptual Design Class* capability gap addresses the lack of robust methodologies and toolsets to support concept development and early systems engineering required in the formulation, evaluation, and documentation of concepts and products to support stakeholder requirements. A critical component of the *Conceptual Design Class* is the creation of standard processes to gather, vet, and refine stakeholder requirements. This includes methods and data repositories for documenting stakeholders needs, shortfalls, gaps, and lessons-learned. In addition mechanisms to perform requirements analysis and functional analysis to identify and quantify measures of effectiveness and performance are needed. These metrics are a necessary precursor to Conceptual Design, Exploration, and Refinement.

Conceptual Design, Exploration, and Refinement is the iterative process of increasing fidelity to define potential solutions with achievable performance to satisfy the customer needs within funding and schedule constraints. This process includes defining the design trade space, generating and maturing concepts, assessing technology readiness, and documenting the concepts. The trade space is defined using knowledge from the functional analysis and allocation activities to bound acceptable functional performance and balance stakeholder needs and project constraints. Alternative concept generation (including advanced technology concept generation) involves determination of the concepts' performance, cost, risk, and employment strategies based on use cases in relevant threat scenarios, creating mission profiles, and generating and executing physics-based engineering models for each concept. Using decision analysis, initial concept screening and potential solution set reduction is performed on the alternative concepts to assess the feasibility of each concept to meet the performance expectations in the use cases. At this point, the remaining candidate concepts are reassessed to ensure that a gap has not appeared in the trade

space; if so, a new concept is introduced to cover the gap. This refined set of remaining concepts is further refined based on higher fidelity engineering analyses, design synthesis, trade studies, effectiveness analyses, and integrated risk analysis procedures to assess performance, effectiveness, risk, cost, and schedule. Quantitative Technology Assessments (QTAs) determine the criticality of the applied technologies and their maturity through identification of the Critical Technology Elements (CTEs). Early assessments of the CTE maturity feeds into programmatic risk control, budgeting for risk reduction, developing the Technology Development Strategy (TDS), and ultimately, Technology Readiness Assessments (TRAs).

Current early-phase acquisition and early system engineering processes are limited in a number of significant ways. Synthesis tools (i.e., tools that estimate the performance of vehicle design concepts based on historical data for similar vehicles) are widely used in both government and industry. Historical data has been shown to be insufficient for evaluation of new, complex, and innovative technologies. The performance of past designs does not address many sources of design uncertainty. Another early-phase limitation is a continued difficulty in estimating vehicle weight and weight distributions of aerospace designs. Key system designs (e.g., airframe structure, engine deck, propulsion systems, etc.) depend on and contribute to vehicle weight. Historical data is not sufficient and first-principal tool sets do not exist. This limitation exists for both government and industry conceptual design groups. Early-phase acquisition is additionally limited by a lack of ability to account for physics coupling (viz., structures, aerodynamics, aero-elastics, thermodynamics and heat transfer, stability, controls, and acoustics) at a level of fidelity necessary to accurately predict responses of vehicle concepts. An ability to synthesize, evaluate, optimize, and assess uncertainty and risk of design concepts during early phase acquisition is critical to ensuring success during the subsequent technology development phase.

3. DESIGN VERIFICATION CLASS

Means for integrated, full-vehicle, full-physics, test and analysis of aircraft designs via simulation methods are needed for the services to verify vehicle performance; perform flight certifications and qualifications; rehearse ground-based and full-scale flight tests; and evaluate planned or potential operational use scenarios all prior to fabrication of test articles, full-scale prototypes, or implementation of aircraft modifications. Such analysis capability is essential for design risk reduction and escaping the traditional design-test-fix paradigm of acquisition. As in conceptual design, existing tool sets that might be used in this role are hindered by a significant lack of physics coupling (viz., structures, aerodynamics, aero-elastics, thermodynamics and heat transfer, stability, controls, and acoustics). High-fidelity tools do exist for most, if not all of the component physics, but they generally address a single physics or design issue. In some cases, existing tools employ mixed-fidelity physics, accurately predicting a principal physics issue, while accounting for the integration of other physics issues with only rudimentary or lower-order models. The domains of appropriate use for existing high-fidelity simulation methods are generally limited to single-physics applications (e.g., aerodynamics only, or structural dynamics only). Full-vehicle design verification requires multi-physics coupling.

4. DESIGN ENVIRONMENT CLASS

An environment that facilitates the application of CBE compute resources throughout the spectrum of acquisition engineering processes is needed. Although it is true that aircraft manufacturers do use CBE in conceptual design and subsequent engineering processes, the relevancy of CBE to existing government acquisition processes is limited to late-phase processes only. Existing environments for aircraft manufacturers also limits, in many respects, the potential of CBE to impact industry design, test, and evaluation processes. Both government and industry processes are limited by computational power that is available to maximize the value of emerging software tools. Paradigms to provide the acquisition community access to necessary CBE compute resources either do not exist, or have not been demonstrated. Infrastructure tool sets necessary for transitioning design data between phases of acquisition or between design states generally do not exist. When they do exist, they usually require inordinate degrees of user expertise, human resources, and calendar time. Problem setup tasks and a lack of robust case management software represents very serious impediments to the effective use of CBE in acquisition.

5. CONCLUSIONS

The CREATE-AV Project will develop and deploy four CBE software toolsets to the defense acquisition engineering community for air vehicles. These are *DaVinci*, *Kestrel*, *Helios*, and *Airframe-Propulsion Integration*. An element of the project known as "*Shadow-Ops*" has been created to establish lines of communication between project development teams and targeted organizations within the defense acquisition engineering community, and to provide important quality controls for project CBE software products. Each of these important project elements is briefly summarized in subsequent papers.

REFERENCES

1. D.E. Post, S. Arevalo, C. Atwood, P. Bell, T.D. Blacker, S. Dey, D., Fisher, D.A. Fisher, P. Genalis, J. Gorski, A. Harris, K. Hill, M. Hurwitz, R.P. Kendall, R.L. Meakin, S. Morton, E.T. Moyer, R. Strawn, D. van Veldhuizen, L.G. Votta, S. Wynn, and G. Zelinski. (2008). A new DoD initiative: the Computational Research and Engineering Acquisition Tools and Environments (CREATE), *J. Phys.: Conf. Ser.* ***125*** *012090*, SciDAC 2008, Seattle, WA

Computationally Based Engineering for Air Vehicle Acquisition: Conceptual Design

Gregory L. Roth *

** U.S. Air Force Aeronautical Systems Center, Wright-Patterson AFB, OH 45433, USA (Gregory.Roth@wpafb.af.mil)*

1. INTRODUCTION

The DaVinci Product

DaVinci is a CREATE-AV CBE software product defined in direct response to the Conceptual Design class of gaps. In January of 2009, the Air Force Materiel Command, Aeronautical Systems Center, Capabilities Integration Directorate (ASC/XRE) at Wright-Patterson Air Force Base in Dayton, Ohio was chosen to host the DaVinci development team. The balance of FY2009 will be used for detailed product planning and initiation of the necessary team-build. DaVinci product planning is a collaborative effort between the Army, Navy, and Air Force organizations having responsibility for defense conceptual design processes.

The DaVinci product will consist of an architecture and framework that provides an open collaborative environment for early air vehicle systems engineering. The intent is to provide unified life-cycle, engineering model-centric, data persistence and encompass functional analysis and allocation, alternative concept generation, trade-space exploration, and acquisition planning. The DaVinci framework will be populated with a set of software support elements. These will include, for example, engineering tools that enable PERCS analysis (Performance, Effectiveness, Risk, Cost, and Schedule) and requirements and technology impact trade studies. In addition, DaVinci will include a key software element that allows the conceptual designer to quickly and intuitively build parameterized, associative, attributed models (including outer mold lines, OML, and corresponding layout of internal structure and sub-systems) necessary to feed higher-fidelity analysis tools (e.g., CREATE-AV Products Kestrel and Helios). This represents a key enabling technology – an ability to quickly generate model descriptions suitable for aerodynamic and structural meshing. This capability will facilitate the relevance of Computationally Based Engineering and High Performance Computing to the earliest phases of acquisition, providing decision makers with consistent-fidelity, multi-physics based decision data, including uncertainty analysis, in a timely way. The DaVinci product also represents an important mechanism for communicating model descriptions between phases of acquisition, currently a significant part of the Design Environment class of capability gaps described earlier.

Kestrel – A Fixed Wing Virtual Aircraft Product of the CREATE Program

Scott A. Morton[1], David R. McDaniel[2], David R. Sears[3], Brett Tillman[4], and Todd R. Tuckey[5]

Air Force SEEK EAGLE Office, Eglin AFB, FL 32542, USA
(Tel: 850-882-1432; e-mail:scott.morton@eglin.af.mil)
(Tel: 205-934-2023; e-mail:david.mcdaniel.ctr@eglin.af.mil)
(Tel: 850-882-6263; e-mail:david.sears@eglin.af.mil))
(Tel: 850-882-6239; e-mail:brett.tillman@eglin.af.mil))
(Tel: 850-882-6278; e-mail:todd.tuckey@eglin.af.mil)

Abstract: This paper documents a new integrating product that allows cross-over between simulation of aerodynamics, dynamic stability and control, structures, propulsion, and store separation. The *Kestrel* software product is an integrating product written in modular form with a Python infrastructure to allow growth to additional capabilities as needed. Computational efficiency will also be improved by targeting the next generation peta-flop architectures envisioned for the 2010+ time frame. *Kestrel* is also targeted to the need of simulating multi-disciplinary physics such as fluid-structure interactions, inclusion of propulsion effects, moving control surfaces, and coupled flight control systems. The *Kestrel* software product is to address these needs for fixed-wing aircraft in flight regimes ranging from subsonic through supersonic flight, including maneuvers, multi-aircraft configurations, and operational conditions. Preliminary results of the F-16C and the F-22 with comparison to experiments are provided. Parallel scalability analysis of *Kestrel* is also included.

Keywords: Aircraft Aerodynamics, Maneuvering Aircraft, Aeroelasticity, Control.

1. INTRODUCTION

Department of Defense High Performance Computing Modernization Program (DoD HPCMP) submitted a POM08 initiative to improve DoD acquisition program timeline, cost, and performance through the use of Computational Science and Engineering (CSE) tools for ships, aircraft, and antenna design and analysis. The resulting program is called the Computational Research and Engineering Acquisition Tools and Environments (CREATE) Program[1] and is managed by Dr Douglass Post of the DoD HPCMP with oversight by the Deputy Undersecretary of Defense, Science and Technology, Dr Andre Von Tillborg. The CREATE Program is a $360M 12 year program executed by a tri-service team under the direction of Dr Post. The air vehicle portion of CREATE is referred to as CREATE-AV and is headed by Dr Robert Meakin of the DoD HPCMP.

Although funding did not begin until October 2007, CREATE-AV planning began in earnest during the summer of 2007. One of the first elements of CREATE-AV to be put in place was a requirements gathering process that would result in software products with a realistic expectancy of impacting the aircraft acquisition process from conceptual design through sustainment.

Following the requirements gather process, the CREATE-AV team determined there were four key software products needed by the acquisition engineering workforce that fit within the available budget and accomplishable in the CREATE Program timeline. The four software products are Helios, a virtual helicopter simulation tool, *Kestrel*, a virtual fixed wing aircraft simulation tool, an airframe-propulsion integration simulation tool, and a conceptual design tool. The Helios and *Kestrel* teams are due to release initial versions of their software products in FY09. The

[1] Senior Research Engineer, 201 West Eglin Blvd, Suite 115.
[2] Research Engineer, University of Alabama Birmingham.
[3] Computer Engineer, 201 West Eglin Blvd, Suite 115.
[4] Software Development Engineer, 201 West Eglin Blvd, Suite 115.
[5] Computer Engineer, 201 West Eglin Blvd, Suite 115.

airframe-propulsion integration and conceptual design software development teams are currently in the process of forming their teams and developing their software product definition.

The *Kestrel* software product is a modularized multidisciplinary fixed wing virtual aircraft simulation tool incorporating aerodynamics, structural dynamics, kinematics, and kinetics. The first version of *Kestrel* is targeted to subsonic, transonic, and supersonic flight conditions with three major capabilities of a single static grid simulation, a single grid rigid body motion simulation, and a deforming single grid aeroelastic simulation.

There are several over-arching design philosophies being applied to CREATE software products. The first is scalability on next generation machine architectures with linear scalability targets for *Kestrel* on the order of 10^4. Another CREATE design philosophy is a "legacy to native" software development approach with near term legacy software being rewritten or factored in to native software over the life of the program. Another important CREATE software design philosophy is modularity. A common architecture in CREATE-AV is a Python based infrastructure and executive and either C or Fortran components. This allows a build-up approach to adding capability and multi-disciplinary physics. It also allows a factored approach to the software, aiding in code maintenance and supportability. One of the most important CREATE design philosophies is to follow a professional software development process incorporating configuration management, automatic unit, integration, and system testing, and user support. To have the desired impact on the DoD acquisition processes, the software has to be maintainable through the life of the program. All CREATE-AV products will have strict version control and configuration management through a Subversion (SVN) repository and continuous integration through nightly unit, integration, and system testing of current versions. User support will have both internet issue tracking and multi-layer live customer support. The following sections present the design and implementation schedule of the *Kestrel* fixed wing virtual aircraft product.

2. *KESTREL* SOFTWARE ARCHITECTURE

The *Kestrel* architecture is a blend of the CREATE design philosophies discussed above. It is a modular approach factoring traditional monolithic solvers into the *Kestrel* Infrastructure and Executive (KIE) piece, components to perform fluid dynamic, structures, kinematics and kinetics and other analysis, and the *Kestrel* User Interface (KUI). Figure 1 depicts a notional view of the *Kestrel* software architecture. The infrastructure and executive is an event-driven Python infrastructure that is component unaware. The components themselves can produce or respond to events and subscribe to or publish data. This allows the infrastructure and executive to be coded once and the eXtensible Markup Language (XML) input file to specify the use case and contributing components. The inputs to KIE are read in from an XML file generated by the KUI. Efficient data handling by KIE is accomplished by passing pointers to "heavy-weight" data or scalars. The resulting overhead was measured at less than 1% compared to a monolithic solver.

In Figure 1 there are two dashed boxes surrounding the components. The left-hand box denotes those components that are shared objects with the KIE and maintained by the *Kestrel* development team. The right-hand box represents executables from outside sources that will exchange data via an executable-to-executable communication path. This feature will be implemented in later versions of *Kestrel* and is intended to allow industry or commodity software to work with *Kestrel* without significant rewrites of their software. An example use of this feature is to allow a commodity CFD solver to be used with all of the other components in *Kestrel*. Another example use would be to incorporate a "blackbox" autopilot from another contractor into the simulation.

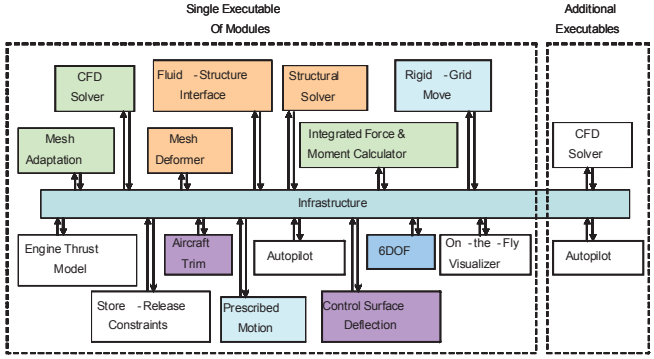


Figure 1: *Kestrel* Architectural Design.

3. CAPABILITY RELEASE SCHEDULE

The development schedule of *Kestrel* is broken up into spirals and phases under each spiral. The two spirals planned concentrate on capability and then performance. In spiral one, there are four releases of *Kestrel* corresponding to four phases that incrementally grow the capability. In spiral two, parallel performance on next generation machine architectures and conversion of legacy code to native code is planned. However, it should be noted that code conversion and performance improvements will occur through all versions. Below is a bulleted list of capabilities.

Spiral One Phase One Capability – *Kestrel* v1.0 currently in beta test

- **Static Rigid Body Aircraft**
  - Single Grid
  - Steady, Unsteady Flow
  - Inviscid, Viscous, Laminar, or Turbulent Flow
- **Dynamic Rigid Body Aircraft**
  - Single Grid
  - Rigid Body Motion
  - Arbitrary Motion file built externally
  - Pitch, Yaw, Roll, Sinusoid or constant rate and hold in KUI
    - body axis and stability axis
- **Flexible Aircraft**
  - Static Position Single Grid Aeroelastic
  - 2nd Order Temporal Coupling
  - Structural Modal Solver
    - Structural modal solver and mode shape forced motion
  - Automated fluid-structure interface couplings and interpolations
  - Algebraic Mesh Deform with 3-4 method choices (Surface Influence, Delaunay, Hybrid)

Spiral One Phase Two – *Kestrel* v2.0

- **Static Rigid Body Aircraft & Captive Store**
  - Single Grid Control Surface Move, Multiple Grids
- **Dynamic Rigid Body Aircraft & Captive Store**
  - Single Grid with 6DOF Motion
- **Static Rigid Body Aircraft & Dynamic Store**
  - Multiple Grids with Prescribed/6DoF Store Motion
- **Flexible Aircraft**
  - Flexible Aircraft and Captive Store Motion

Spiral One Phase Three – *Kestrel* v3.0

- **Dynamic Rigid Body Aircraft & Captive Store**
  - Single Grid with FCS, 6DOF Store Motion
  - Single Grid with Dynamic Propulsion, FCS, 6DOF, and Trim
- **Static Rigid Body Aircraft & Dynamic Store**
  - Multiple Grids with 6DOF Store Motion
  - Multiple Grids with 6DOF Store Control Surfaces

Spiral One Phase Four – *Kestrel* v4.0

- **Dynamic Rigid Body Aircraft & Captive Store**
  - Multiple Grids with Dynamic Propulsion, Flight Control System, 6DOF, and Trim
- **Dynamic Rigid Body Aircraft & Dynamic Store**
  - Multiple Grids Prescribed Aircraft Motion, and 6DOF Stores
  - Multiple Grids, 6DOF Aircraft Motion, and 6DOF Stores
- **Flexible Aircraft**
  - Dynamic Flexible Aircraft and Captive Stores
  - Dynamic Flexible Aircraft and Dynamic Stores

The current status of *Kestrel* is the following: a) *Kestrel* v1.0 is in beta testing with all three major capabilities working, b) *Kestrel* is compiled and running on four of the DoD HPCMP Defense Shared Resource Centers (DSRC) supercomputers--Hawk and Falcon at the AFRL DSRC, Jaws at the Maui High Performance Computing Center DSRC, and MJM at the Army Research Laboratory DSRC, and c) *Kestrel* v2.0 has passed the CCB and in design.

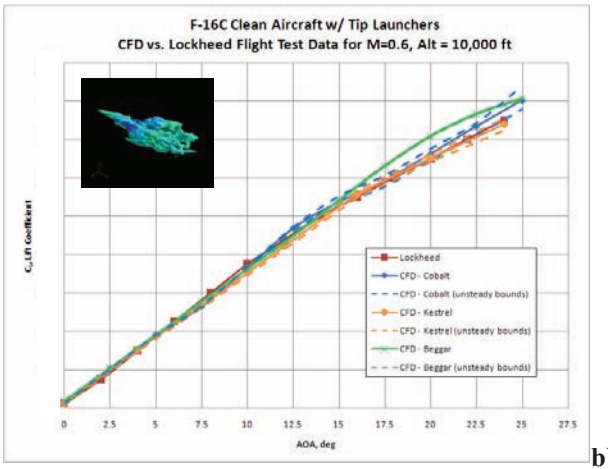
4. PRELIMINARY RESULTS

This section presents preliminary solutions computed with *Kestrel* with comparison to experiments and other solvers in Figures 2 a) and b). In addition to the comparisons, parallel scalability for *Kestrel* is included in Figures 3 a) and b).

Figure 2 a) depicts the lift coefficient versus angle of attack for the F-16C in a clean configuration for Lockheed flight test data and *Kestrel*, Cobalt, and Beggar computational fluid dynamics data. Solutions are presented for a Mach number of 0.6, an altitude of 10,000 ft, and a range of angles of attack between 0 and 24 degrees. The *Kestrel* solutions virtually lay on top of the flight test data and demonstrate an increased variation between the mean solution and maximum and minimum values of lift for increased alpha. *Kestrel* and Cobalt also both demonstrate a "knee in the curve" near 12 degrees alpha consistent with the flight test data. Figure 2 b) depicts the drag coefficient versus angle of attack for *Kestrel*, Cobalt, and Beggar. No flight test data was available for drag coefficient.

Figure 3 a) and b) depict the parallel scalability performance of *Kestrel* for a 6 million cell mesh on various platforms. These results are for *Kestrel* with no performance tuning and represent a baseline for improvement in the future. The *Kestrel* parallel scalability as measured by efficiency is at 70% for this fairly small mesh on 1024 processors. Efficiency in this case is defined by comparing to the linear parallel scalability for an increase in processor count. Achieving the *Kestrel* target of 90% efficiency on 1024 processors for a 20 million cell mesh has a high likelihood of success due to the baseline scalability observed on a 6 million cell mesh.

a)



b)

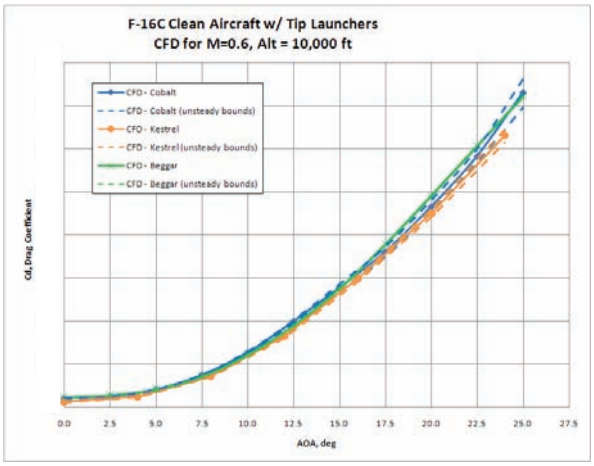


Figure 2: F-16C at a Mach number of 0.6 and an altitude of 10,000ft: a) lift coefficient versus angle of attack comparison of *Kestrel* with flight test data from Lockheed and two other solvers (Cobalt and Beggar), b) drag coefficient versus angle of attack comparison of *Kestrel* with two other solvers (Cobalt and Beggar).

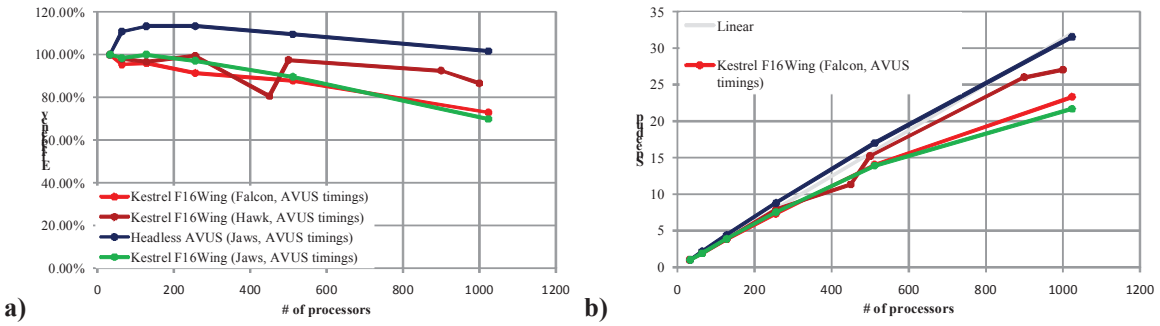


Figure 3: *Kestrel* Parallel Speedup for a 6 million cell mesh on three different DoD parallel computers.

4. CONCLUSIONS

In conclusion, significant progress has been made since the inception of the CREATE Program on 1 October, 2007. In CREATE-AV, requirements have been gathered, refined, and prioritized, software capabilities to meet the requirements have been defined, four software products have been selected to meet user priorities and the available budget, and two of the four software product teams are near a release of their first version. The *Kestrel* software product has been developed following the over-arching design philosophies of the CREATE Program and to meet the specific needs of the aircraft acquisition engineer and scientist communities. The *Kestrel* product has been developed from the ground up following sound software development practices and innovative solutions to the user requirements have resulted from the process. An intensive validation and verification effort is currently underway to ensure *Kestrel* is suitable for production use by the acquisition community.

ACKNOWLEDGEMENTS

Material presented in this paper is a product of the CREATE-AV Element of the Computational Research and Engineering for Acquisition Tools and Environments (CREATE) Program sponsored by the U.S. Department of Defense HPC Modernization Program Office. The authors would also like to acknowledge the Air Force SEEK EAGLE Office and 46 Test Wing RANG/VTSO for their facility and financial management support.

REFERENCES

[i] D. E. Post, S. Arevalo, C. Atwood, P. Bell, T. D. Blacker, S. Dey, D. Fisher, D. A. Fisher, P. Genalis, J. Gorski, A. Harris, K. Hill, M. Hurwitz, R. P. Kendall, R. L. Meakin, S. Morton, E. T. Moyer, R. Strawn, D. van Veldhuizen, L. G. Votta, S. Wynn, and G. Zelinski, Journal of Physics: Conference Series vol. 125 (2008) 012090.

Computationally Based Engineering for Air Vehicle Acquisition: Airframe-Propulsion Integration

Robert H. Nichols *

** University of Alabama Birmingham, Birmingham, Alabama 35294, USA*
(Robert.Nichols@arnold.af.mil)

1. INTRODUCTION

The Airframe-Propulsion Integration Product

The A-PI software product is being architected as a module to be driven either by Kestrel or Helios to facilitate integrated evaluation airframe/engine-inlet/engine-fan compatibility and nozzle/after-body interactions, including nozzle transients, at any point in the design cycle. In October of 2009 the Air Force Arnold Engineering Development Center was chosen to host the A-PI development team at their facility in Tullahoma, TN. FY2009 is a detail planning and requirements gathering year with a preliminary low-order propulsion module scheduled for delivery to the Kestrel team.

Motivation for a capability for airframe propulsion integration is driven by the fact that current design practice does not allow integrated testing of air vehicle concepts until very late-phase acquisition (i.e., fabrication of full-scale prototypes), creating design uncertainty and potential for acquisition costs due to late discovery of performance anomalies and associated re-design cycles. For example,

i. Spillage drag is currently determined in the wind tunnel by varying the mass flow through a model of the inlet design. Effects of throttle transients are not included in wind tunnel simulations, so the current design methodology does not address unsteady flow effects on external surfaces of an aircraft that often occurs with rapid throttle transients. The detrimental effects of the resulting unsteady flow are often not uncovered until full scale flight tests.

ii. Current industry practice for integrating the inlet and engine is for the airframe and engine original equipment manufacturers (OEMs) to agree on a level of distortion to be provided by the inlet and accommodated by the engine. The distortion level is determined in subscale wind tunnel tests. Distortion simulation screens are developed from the wind tunnel results and placed in front of the engine to determine if the patterns are acceptable in terms of performance and operability. The inlet and engine are not evaluated together until they are both integrated into the airframe for flight tests.

iii. Nozzle/after-body integration is currently performed in subscale wind tunnel tests using perfect gas jets to simulate the engine exhaust. The results are then corrected using empirical relationships to account for real gas and temperature effects. The first real coupling of the engine and airframe again occurs in flight tests. Effects of nozzle transient effects on the external flow are also not simulated until flight tests.

As noted earlier, the CREATE-AV Airframe-Propulsion Integration product is being architected as a module to be executed in both Kestrel and Helios. The architecture assumes a mature state that includes high fidelity simulation of engine inlet through exhaust and all physics in between. However, it is recognized that the scope and resources of the CREATE-AV project alone are insufficient for complete fulfillment of this end-state. Accordingly, product development will approach this end-state by degrees, delivering capability to the targeted acquisition engineering workforce in response to prioritized requirements. A progression of capability might include the following capabilities.

i. Module that facilitates incorporation of 0-D and 1-D engine models into Kestrel and Helios.

ii. Add capability to the module for Navier-Stokes simulation of engine inlet and rotating machinery, including aero-elastic effects of fan blades. Add capability to directly represent nozzle geometry, including moving walls, and multi-species capability to the solver to correctly represent different gasses of exhaust jet. Employ engineering model for all other components and processes of the engine.

iii. Add capability to the flow solver for accommodation of full non-equilibrium chemistry to account for the reactions associated with combustion. Employ an engineering model of multi-phase aspects of the combustion process.

iv. Add capability to the flow solver for accommodation of multi-phase flow associated with transition of jet-fuel droplets into gasses. This capability is important in accurate prediction of combustor and afterburner performance.

Progressions 1 and 2 above are the planned contributions of the CREATE-AV Project toward integration of engine and airframe into system analysis capability. It is possible that the project could reach progression iii. Given the complexity of the physics and current state-of-the-art for propulsion and combustion modeling, progression iv is clearly beyond the scope of CREATE.

Computationally Based Engineering for Air Vehicle Acquisition: Rotary Wing Simulation

Venkateswaran Sankaran *

** US Army Research, Development, and Engineering Command, AFDD, Moffett Field, CA 94035, USA (vsankaran@merlin.arc.nasa.gov)*

1. INTRODUCTION

The Helios Product

The genesis of the Helios product name is a hybrid word derived from Helicopter Overset Solutions, or "HeliOS". The OS acronym has since been dropped and the product simply goes by the name "Helios". Helios is actually the product of the HPC Institute for Advanced Rotorcraft Modeling and Simulation (HIARMS), sponsored by the DoD HPC Modernization Program Office and operated by the Army AMRDEC at Moffett Field, California. The strategic goals of the HIARMS are closely aligned with those of the CREATE-AV Project. This is true to such an extent that Helios developers are now full participants in CREATE-AV Project management and product development processes, though funding lines are separate. Helios development began in FY2006 and will continue through the end of FY2011. After this time, development will continue with the combined support of the Army host of the Helios development team and CREATE-AV Project to directly address CREATE priorities. Helios and Kestrel are complementary software products. Both directly address the Design Verification class of gaps described above. Helios targets rotary-wing air vehicles, while Kestrel targets the fixed-wing complements. The technologies represented in these products overlap, yet each have a number of unique aspects as well. The CREATE-AV Project management exploits this reality to leverage both the expertise of the respective development teams and software components. The relationship is highly beneficial to the project in many ways, but specifically in terms of software development practices relative to API definition, module interoperability, and long-term software maintenance.

At maturity, Helios will facilitate full rotorcraft (all types, manned and unmanned) high-fidelity simulation, including direct simulation of relative motion between rotors and airframe, as well as engineering models of rotor systems; and stores/cargo carriage and release for realistic flight conditions (hover, forward flight, and transition and conversion, for vehicle concepts that employ such technology) and operational conditions such as refueling maneuvers or takeoff and land maneuvers in benign and harsh environments (e.g., pitching/heaving ship decks). Helios provides for the direct coupling of physics – rotor aero-elastic effects (flapping, lead/lag, and torsional), rotor/wake and airframe interactional dynamics, and propulsion effects (e.g., aerodynamics and inlet performance, exhaust re-ingestion during hover, exhaust plume dynamics, etc.). Together, these software attributes allow for early assessment for rotorcraft designs (at preliminary design and final design) with corresponding opportunities for fault detection prior to fabrication of scale models or full-scale prototypes. After production, the capabilities allow for ground-based and flight test rehearsals for test scoping, mission planning, and operational testing.

The Helios product is a high-fidelity multi-disciplinary computational analysis platform for rotorcraft aeromechanics applications. Rotorcraft flow fields are challenging because they are inherently multidisciplinary, requiring the solution of moving-body aerodynamics coupled with structural dynamics for rotor blade deformations and vehicle flight controls. Moreover, the aerodynamics solution is difficult because of the need to resolve multiple spatial and temporal scales in the problem, particularly the accurate propagation of the rotor-blade tip vortices into the far-wake. Unlike their fixed-wing counterparts, rotorcraft fly in their own wake, demanding accurate prediction of these flow physics relatively long distances into the field. Helios handles the aerodynamics solutions in an innovative manner using a dual-mesh paradigm: unstructured meshes in the near-body region and Cartesian meshes in the off-body region. The unstructured near-body meshes enable ease of mesh generation around complex bodies and configurations while, at the same time, preserving accuracy of the viscous scales in the boundary layers. The Cartesian off-body mesh solution, on the other hand, allow the use of efficient data structures, high-order accurate discretizations and ease of mesh adaptation, all of which are crucial for accurately resolving the tip vortices in the far-wake. Moreover, the near-body mesh rotates, moves and deforms with the rotor blades and vehicle, while the off-body meshes are usually stationary. Data transfer between the two mesh systems are performed through the use of overset domain connectivity transfer technology that dynamically cuts holes and fringes in the meshes to accommodate the relative motion between the mesh systems.

Helios utilizes a Python-based infrastructure called the Software Integration Framework (SIF) to couple the modules responsible for the aerodynamics, structural dynamics, vehicle flight dynamics and six degree-of-freedom motions. The modules themselves are derived in most instances from legacy single-discipline computer codes and are written in variety of languages: FORTRAN, C and C++. The use of Python enables the wrapping of these codes into modules or components, the efficient transfer of data between them and their execution in a scalable parallel cluster environment. Importantly, SIF provides a light and flexible infrastructure which accesses the components at a very high-level and requires little customization or rewriting of the underlying legacy (or new) codes. Simultaneously, it also offers the potential for specifying a set of component interfaces that would allow the underlying codes or components to be re-engineered or even changed entirely without interfering with the other codes in the infrastructure.

Presently, Helios is nearing the release of the first version of the software product, code-named Whitney. This first version provides capabilities for stand-alone fuselage aerodynamics, optionally combined with a momentum disk model, and isolated rotor analysis with and without aeroelastic and comprehensive analysis coupling effects. The software product is accompanied by a front-end graphical user-interface, which is based upon a common graphical engine developed by the Kestrel team. This user-interface allows for ease of preparing grids for Helios and for setting up run-time inputs. Like all CREATEAV products, Helios will undergo a beta-certification process by the CREATEAV ShadowOps team prior to actual beta release. Additional functionalities will be introduced in future releases following an approximately annual release cycle.

PARALLEL AND MESHFREE: NEW FRONTIERS OF CFD

Parallel and Meshfree: new frontiers of CFD

L. A. Barba*

**Department of Mechanical Engineering, Boston University, Boston, MA 02215*

Abstract: For the best part of 50 years, advances in algorithms and the increasing computing power have made computational fluid dynamics, CFD, a mature discipline. Which are some remaining frontiers of the discipline? It is common knowledge that one of the challenges in fluid simulation continues to be the need to straddle many scales. New computational methods still need to be developed that are able to adapt to the many scales of a problem. Another frontier recently opened is the development of hardware-aware software. Multi-core computers are on everyone's desktops nowadays, and a growing trend in using graphics cards and other specialized hardware is buzzing. In all of these frontiers, there is great potential for meshfree methods. Particle-type formulations for CFD offer an alternative which is low in numerical diffusion, devoid of numerical dispersion and stability constraints. Also, meshfree methods offer a natural adaptivity in situations where mesh generation is a huge burden (*e.g.* moving boundaries). Meshfree methods could be especially well-suited to exploit the new hardware technologies entering the scene. I will present an overview of some aspects of meshfree simulation in fluid dynamics, and recent progress in algorithms and parallel implementations, including novel hardware.

Keywords: meshfree, particle method, parallel computing, novel hardware

1 INTRODUCTION

In an often cited complaint of computational fluid dynamics practitioners, the generation of a good quality computational mesh is estimated to be maybe 80% of the cost of a simulation cycle. Perhaps this is today an exaggeration, as the quote has been repeated for years without editing the estimate. Or perhaps not. As computing power has grown exponentially, more and more complex problems are attempted by the community. Presently, many of the cutting-edge results in fluids simulation involve highly irregular immersed shapes or fluid conduits (*e.g.*, in biological fluid dynamics applications), or moving boundaries. In these types of problems not only the generation but the subsequent maintenance of a good mesh can be an onerous part of the job.

Before we begin discussing some recent research progress in the development and implementation of meshfree methods, maybe it is a good idea to agree on some basic terminology. What is a *mesh*? First, it is a means by which a continuum problem is translated into a finite description—but of course meshfree method require this, too. Second, it is a geometric representation of the object(s) that is(are) to be analyzed by the CFD program. Again, meshfree methods need something like this also. What characterizes the mesh-based approach in particular is that the above are provided by a set of nodes and the interconnectivity of these nodes, which has to be maintained. Meshfree approaches aim at utilizing the set of nodes but doing away with the connectivity and its maintenance over time. The simplification that this introduces in the geometric representation of objects and surfaces is substantial.

During the celebration of the 75th anniversary of the Graduate Aeronautical Laboratories in Caltech (November 2003), I had the privilege of meeting Professor Milton Van Dyke[1]. When he asked me what I was working on, and I explained about the efforts to simulate fluid flow without using a mesh, he replied: "It sounds like magic." Today, a critical mass of researchers are demonstrating that magic in a variety of applications, including both fluids simulation and computational mechanics. The list of variations that have been under development is dizzying: from the oldest vortex particle method [4] and smooth particle hydrodynamics method, SPH [11], to diffuse element method [16], element-free Galerkin [3], reproducing kernel particle method, RKPM [15, 14], *hp*-clouds [9, 10], and a half-dozen more.

It would be too ambitious to try to give a real overview of meshfree methods on this occasion. Instead, I will concentrate on three topics. First, how radial basis function (RBF) methods have been combined with the vortex particle method to produce a fully meshfree system of direct numerical simulation. Second, the importance of clever algorithms to provide efficiency and ensure that these methods are competitive with mainstream CFD schemes—I will focus on

[1] Author of the classic book "An Album of Fluid Motion".

the fast multipole method as a paradigm of $\mathcal{O}(N)$ methods with multi-resolution capability. Third, I will discuss innovations for using new hardware to implement numerical algorithms. This last topic will bring up the possibility that meshfree and particle methods can take better advantage of the massively parallel hardware, such as graphic card accelerators.

2 FULLY MESHFREE VORTEX PARTICLE METHOD FOR FLUID SIMULATION

The vortex particle method is used to simulate incompressible fluid flow based on the vorticity formulation of the Navier-Stokes equation. A particle-like approximation is used on the vorticity field $\omega = \nabla \times u$, as follows,

$$\omega(x,t) \approx \omega_\sigma(x,t) = \sum_{i=1}^{N} \gamma_i \zeta_\sigma \left(x - x_i(t)\right). \tag{1}$$

Here, ζ_σ is the particle basis function, γ_i are the particle weights, and the scattered particles located at the points x_i are convected by the fluid velocity u to satisfy vorticity transport. In this Lagrangian formulation, there is an incremental loss of spatial accuracy as the x_i become disordered. This effect is not catastrophic, but it can quickly bring accuracy down to unacceptable levels. The reason is that as the vortex particles follow the flow strain, they cluster together in some areas and open gaps in other areas, and so they are unable to represent a continuous field. This, added to the fact that convergence proofs for the vortex method relied on the assumption of particle overlap—$h/\sigma < 1$ for h the average particle separation and σ their length scale—, meant that only short-time calculations were advised. This situation changed when an interpolation scheme for particle weights was first introduced [13] in what is now called 'particle redistribution' or 'remeshing'. Particle remeshing allowed for the first time long simulations of high-Reynolds number flow with the vortex method. However, it introduced back the mesh into the numerical system, and with it numerical diffusion and issues related to accommodating irregular boundaries. Nevertheless, many excellent results were produced using vortex methods with remeshing, such as [17, 6].

A fully meshfree alternative, in the sense that nodes can be scattered and no regularity requirement is introduced in the formulation, was introduced in [2], where re-location of the particles is accomplished via radial basis function interpolation. When a function f is only known at scattered points $X = \{x_1, \cdots, x_N\}$, it can be approximated by a linear superposition of radial basis functions (RBFs), $\phi_i = \phi(x - x_i)$, as,

$$f(x) \approx f_\phi(x) = \sum_{i=1}^{N} \alpha_i \phi(x - x_i) \tag{2}$$

The parallel between (2) and (1) is obvious. The crux of the RBFmethod is finding the appropriate values of the weights, α_i, so that the representation (2) approximates well the function f. To find the weights, one uses the known values of the function at the data locations, $f(x_j)$ with $j = 1 \cdots N$, and solves a collocation problem:

$$f(x_j) = \sum_{i=1}^{N} \alpha_i \phi(x_j - x_i) \qquad \text{or} \quad \vec{f} = \Phi_{ij}\vec{\alpha} \tag{3}$$

Solving such a system can be computationally expensive—$\mathcal{O}(N^2)$ if using an iterative method, due to the matrix-vector multiplications with a dense matrix—and is moreover complicated by the fact that the matrix Φ_{ij} is ill-conditioned for most choices of ϕ. Compact support basis functions have been developed to deal with these issues, but sacrifice accuracy and convergence rate, in exchange for computational efficiency.

3 COMPUTATIONAL CHALLENGES

Solving the RBF collocation problem, as described above, involves a linear system with a fully-populated and ill-conditioned matrix. Clearly this is a challenge when N is large. Another classic challenge for particle methods is the calculation of pair-wise interactions, in the case of the vortex method, to obtain the velocity of the particles from the vorticity. Using the Biot-Savart law of vorticity dynamics, the velocity of the particles is given by,

$$u(x_j) = \sum_{i=1}^{N} \gamma_i \mathbb{K}_\sigma(x_j - x_i) \tag{4}$$

where the kernel $\mathbb{K}_\sigma$ is related to the particle basis function ζ_σ via the Biot-Savart integral. A common choice for the particle basis function is a Gaussian:

$$\zeta_\sigma(x) = \frac{1}{2\pi\sigma^2}\exp\left(\frac{-|x|^2}{2\sigma^2}\right), \qquad \text{and} \qquad \mathbb{K}_\sigma(x) = \frac{1}{2\pi|x|^2}(-x_2, x_1)\left(1-\exp\left(-\frac{|x|^2}{2\sigma^2}\right)\right). \tag{5}$$

For this choice, the kernel $\mathbb{K}_\sigma$ does not decay fast enough that it can be neglected in the far field, and thus the evaluation of the velocity (4) requires $\mathcal{O}(N^2)$ operations. This was for quite some time an impediment to the adoption of the particle method, as it was simply not competitive compared with mainstream CFD schemes.

Two types of methods have been developed to more efficiently compute the particle interactions. One relies on interpolating the particle information onto a mesh and solving for the velocity field efficiently on the mesh, to then interpolate the computed field back to the particle locations. This approach is known as the vortex-in-cell method [7, 5]. As in the process of particle remeshing described in the previous section, the vortex-in-cell calculations introduce a mesh into the numerical system. The second method for the efficient computation of the particle interactions relies on a hierarchical decomposition of space, to identify what is 'near' and what is 'far' from an evaluation point, and applies series expansions to represent clusters of particles in the far domain; it is essentially meshfree. The archetype of this second type of method is the Fast Multipole Method, FMM [12].

The FMM algorithm can be rather complex to fully grasp, and difficult to program, which has been a barrier for adoption and arguably diminished its impact. Moreover, parallelization can be quite tricky. To aid in a more widespread adoption of the FMM, we have recently developed PetFMM[2], a parallel fast multipole library implemented within the framework of PETSc[1]. The current version consists of a complete implementation of the FMM in parallel, with the feature of providing load balancing automatically via an optimization approach [8]. A prominent feature of this software is that it is designed to be extensible—the goal is to effectively unify efforts involving several algorithms which are based on the same principles of the FMM. This should be a significant contribution to the meshfree and particle methods community.

Going back now to the problem of solving the ill-conditioned RBF interpolation problem, we also have some significant progress to report. In [18] we developed a domain decomposition approach with an added mean field, designed to take advantage of the rapid decay of the Gaussian basis function. The algorithm was shown to have excellent algorithmic efficiency, converging rapidly to machine precision, and an $\mathcal{O}(N)$ complexity and storage requirement. At this time, the research code for this fast RBF method is available from Google Code[3], but we are initiating the development of a parallel version which will be shared in an open source framework similar to the PetFMM software.

4 HARDWARE ACCELERATION

A new wave in scientific computing has the community buzzing: the use of graphics processors for programmable tasks, with the aim of exploiting their massively parallel architecture. The huge attention that GPUs are attracting stem from the large computing capacity and the low cost of these devices. When GPUs are used as accelerators for CPUs, the hardware configuration is often referred to as a hybrid system. The performance achieved from hybrid systems far exceeds that of today's best X86 processor. Therefore, the potential for order of magnitude speed-ups is enticing many venturesome researchers to attempt implementation of their algorithms for such systems.

Not all problems are massively parallel, however. Indeed, many algorithms do not map well to the GPU architecture. Often 'porting' a code from a serial version is inefficient and complicated. Rethinking algorithms might be required, so that they are more suitable for the new architecture. Because the GPU has been developed for graphics computations, the hardware itself is very different from a normal computer. Programming for the massively parallel GPU is difficult—the number of parallel executions can easily be in the order of several thousand kernels (a kernel is a small function). Imagine that programming in parallel for clusters of computers is already a challenge!

Our approach to the task of implementing problems for systems with GPU acceleration is to think about them as 'heterogeneous computing' applications. The philosophy of heterogeneous computing is reducing the overall execution time by using different types of hardware available within the same system. Commonly, algorithms do not fit only a single model (they can have some parts that require sequential execution, and others that are more suitable for parallel execution), so identifying and exposing parallelism in an algorithm is the first step. Often, the sequential algorithm will need to be re-thought. As a result, it is possible that a less 'efficient' but more parallel algorithm is produced.

[2] PetFMMstands for 'portable extensible toolkit for FMM', as in PETSc, which is 'portable extensible toolkit for scientific computing'.
[3] The code is found at `http://code.google.com/p/pyrbf/`.

For example, in an N-body simulation, where all the particles in the system interact with each other, if the interactions are symmetric—the interaction (a, b) is equal to the interaction (b, a)—a sequential code can take advantage of the symmetry to reduce the operations. In the GPU, the serial optimization based on symmetry is actually more complicated to implement and could also be slower than computing all the interactions directly!

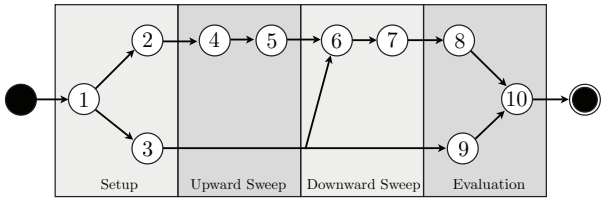


Figure 1. The tasks in the DAG are: ① Tree construction ② Particle clustering ③ Listing of cluster interactions ④ Particle to multipole calculations ⑤ Multipole to multipole translations ⑥ Multipole to local transformation ⑦ Local to local translation ⑧ Local to particle evaluation ⑨ Near domain evaluation ⑩ Adding near and far field contributions.

As mentioned above, the first step is exposing parallelism in the algorithm. In the second step, we implement and distribute the code between the CPU and the GPU. Our goal is to efficiently distribute the work, taking advantage of the fact that CPU and GPU could also work in parallel. As an example, consider the FMM algorithm. If we look at the FMM from a sequential point of view we have four sequential stages: setup, upward sweep, downward sweep, and evaluation. But the algorithm can be further analyzed for more parallelism. We use a Directed Acyclic Graph (DAG) representation of the algorithm to express the finer-parallelism between tasks, as shown in Figure 1. This tool makes distributing the work evenly between the CPU and GPU a more manageable task.

The approach to exploitation of novel hardware just described is currently being used to produce a GPU implementation of the FMM algorithm; this is work in progress. But it also has revealed the following: due to the fact that particle methods are characterized by a higher computational intensity, they have greater potential to take advantage of the massively parallel hardware. In contrast, mainstream mesh-based CFD methods, such as finite difference, are not computationally intensive, and will have a harder time exploiting the novel hardware paradigm introduced by GPUs.

5 CONCLUDING REMARKS

Meshfree methods have been gaining increasing levels of interest in recent years. In computational fluid dynamics, so far the most prevalent meshfree methods are the vortex particle method and smooth particle hydrodynamics methods. An impediment to the wider adoption of these methods for quite some time was the perceived computational expense, in comparison with more mainstream CFD schemes. Currently, efficient algorithms are becoming available that are able to make meshfree methods competitive in terms of computational complexity. Recent progress includes the development of fast solution algorithms for the Gaussian radial basis functions used in vortex methods (in SPH it has become customary to use compact support basis functions, but potentially Gaussians could be used instead with the new efficient algorithms). Drives to produce software libraries to aid in the adoption of meshfree methods, aggregating the experience of many researchers through the open source model, will also contribute to the maturation of the field. As this happens, we will be able to explore with meshfree methods some of the remaining frontiers of CFD:

- The need to straddle many scales in a simulation—particles of variable scale and density can be used quite naturally to resolve different scales in the computational domain.
- Algorithms that are able to detect and adapt to a solution—meshfree methods naturally can adapt. Means of detecting features in a solution remain a challenge.
- Hardware-aware software—meshfree methods could be especially well-suited to exploit the emerging massively parallel architectures, such as GPU.
- The capacity to tackle problems wit complex/moving geometry—enforcing boundary conditions with meshfree methods is still an area where ample progress needs to be made. As this progress is made, the new formulations will be easily extended to complex/moving boundaries, due to the simplified geometry descriptions allowed by the meshfree model.

REFERENCES

[1] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. Curfman-McInnes, B. F. Smith, and H. Zhang. PETSc User's Manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2002.

[2] L. A. Barba, A. Leonard, and C. B. Allen. Advances in viscous vortex methods — meshless spatial adaption based on radial basis function interpolation. *Int. J. Num. Meth. Fluids*, 47(5):387–421, 2005.

[3] Ted Belytschko, Y. Y. Lu, and L. Gu. Element-free Galerkin methods. *Int. J. Num. Meth. Eng.*, 37(2):229–256, 1994.

[4] A. J. Chorin. Numerical study of slightly viscous flow. *J. Fluid Mech.*, 57:785–796, 1973.

[5] G.-H. Cottet. Convergence of a vortex-in-cell method for the two-dimensional Euler equations. *Math. Comp.*, 49(180):407–425, 1987.

[6] G.-H. Cottet and P. Poncet. Particle methods for direct numerical simulations of three-dimensional wakes. *J. Turbulence*, 3:038, 2002.

[7] B. Couet, O. Buneman, and A. Leonard. Simulation of three-dimensional incompressible flows with a vortex-in-cell method. *J. Comp. Phys.*, 39(2):305–328, 1981.

[8] Felipe A. Cruz, L. A. Barba, and Matthew G. Knepley. PetFMM — a dynamically load-balancing parallel fast multipole library. To be submitted; preprint available `http://people.bu.edu/labarba/pubs/CruzBarbaKnepley2009.pdf`, 2009.

[9] C. A. Duarte and J. Tinsley Oden. An h-p adaptive method using clouds. *Comp. Meth. Appl. Mech. Engrg.*, 139(1–4):237–262, 1996.

[10] C. A. Duarte and J. Tinsley Oden. h-p clouds—an h-p meshless method. *Numer. Meth. Partial Diff. Eq.*, 12(6):673–705, 1996.

[11] R.A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Mon. Notices R. Astron. Soc.*, 181:375–389, 1977.

[12] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.

[13] P. Koumoutsakos and A. Leonard. High-resolution simulations of the flow around an impulsively started cylinder using vortex methods. *J. Fluid Mech.*, 296:1–38, 1995.

[14] Wing Kam Liu, Yijung Chen, R. Aziz Uras, and Chin Tang Chang. Generalized multiple scale reproducing kernel particle methods. *Comp. Meth. Appl. Mech. Engrg.*, 139(1–4):91–157, 1996.

[15] Wing Kam Liu, Sukky Jun, Shaofan Li, Jonathan Adee, and Ted Belytschko. Reproducing kernel particle methods for structural dynamics. *Int. J. Num. Meth. Eng.*, 38(10):1655–1679, 1995.

[16] B. Nayroles, G. Touzot, and P. Villon. Generalizing the finite element method: diffuse approximation and diffuse elements. *Comp. Mech.*, 10(5):307–318, 1992.

[17] P. Ploumhans, G. S. Winckelmans, J. K. Salmon, A. Leonard, and M. S. Warren. Vortex methods for direct numerical simulation of three-dimensional bluff body flows: Application to the sphere at Re=300, 500 and 1000. *J. Comp. Phys.*, 178:427–463, 2002.

[18] C. E. Torres and L. A. Barba. Fast radial basis function interpolation with Gaussians by localization and iteration. In press: *J. Comput. Phys.*, doi:10.1016/j.jcp.2009.03.007, 2009.

Hybrid OpenMP-MPI Approach for Smoothed Particle Hydrodynamics

C. Moulinec*, R. Issa, D. Latino***, P. Vezolle****, D.R. Emerson*, X.J. Gu***

**STFC Daresbury Laboratory, Warrington, WA4 4AD, UK*
(charles.moulinec@stfc.ac.uk, david.emerson@stfc.ac.uk, xiaojun.gu@stfc.ac.uk)
***EDF R&D, National Hydraulics and Environment Laboratory, quai Watier, 78400 Chatou, F*
(e-mail: reza.issa@edf.fr)
****IBM Middle East, Dubai Internet City, P.O. Box 27242, Dubai, UAE*
(e-mail: dlatino@ae.ibm.com)
*****IBM France, Rue de la Vielle Poste, BP 1021, F-34006, Montpellier, F*
(e-mail: vezolle@ae.ibm.com)

Abstract: 3-D SPH simulations are getting more affordable due to ongoing computer hardware improvement. This work aims to demonstrate that a hybrid OpenMP-MPI approach applied to SPH can enhance the performance of the code on multi-core machines. The case of a dam breaking and impacting an obstacle, where experimental data are available, is used for illustation. The simulation is carried out with 1.3M particles initially regularly distributed. Performance shows that the hybrid OpenMP-MPI approach performs better than MPI only on 1 rack of a BlueGene/P. Comparison with experiments show good agreement in terms of water elevation.

Keywords: SPH, HPC, hybrid OpenMP-MPI.

1. INTRODUCTION

The pseudo-compressible Navier-Stokes equations written in Lagrangian form for Newtonian fluids and incompressible laminar flows read:

$$\begin{cases} \dfrac{D\mathbf{u}}{Dt} = -\dfrac{1}{\rho}\nabla p + \nabla\cdot(\nu\nabla\mathbf{u}) + \mathbf{F^e}, \\ \dfrac{D\rho}{Dt} = \rho\nabla\cdot\mathbf{u}, \end{cases} \tag{1}$$

where $\mathbf{u}$ is the velocity vector, t the time, ρ the density, p the pressure, ν the molecular viscosity and $\mathbf{F^e}$ an external force, such as gravity, or any other force driving the flow. '∇' and '$\nabla\cdot$' are respectively the gradient and divergence operators. The system described by Eqs. (1) is closed by a state equation for the pressure [1], which reads:

$$p = \frac{\rho_0 c_0^2}{\gamma}\left[\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right], \tag{2}$$

where ρ_0 is a reference density, c_0 a numerical speed of sound and γ a constant coefficient.
Particle position $\mathbf{r}$ is updated at each temporal iteration by the following time integration:

$$\frac{d\mathbf{r}}{dt} = \mathbf{u}. \tag{3}$$

Equations (1), (2) and (3) are discretised explicitly in time and the SPH approach is used to perform spatial discretisation.

The key idea of SPH is that any flow property A can be expressed in any point of the fluid domain, localised by $\mathbf{r}$, by a convolution product with the Dirac distribution δ, as:

$$A(\mathbf{r}) = \int_{\Omega} A(\mathbf{r}')\delta(\mathbf{r}-\mathbf{r}')d\mathbf{r}', \tag{4}$$

where Ω is the whole fluid domain. The Dirac distribution δ is approximated by a smoothing function w_h called kernel (h is its smoothing length), as:

$$A(\mathbf{r}) = \int_{\Omega} A(\mathbf{r}')w_h(\mathbf{r} - \mathbf{r}')dr' + O\left(h^2\right). \tag{5}$$

A discrete set of equations is obtained by approximating the integral of Eq. (5) by a Riemann summation, as follows:

$$A(\mathbf{r}) = \sum_b \frac{m_b}{\rho_b} A_b w_h(|\mathbf{r} - \mathbf{r}_b|) + O(h^2), \tag{6}$$

where A_b denotes the value of A for particle b, with mass m_b, located at position $\mathbf{r}_b$. The summation is now discrete and the elemental volume dr' (see Eq. (5)) is calculated as the particle volume defined from the mass and the density as m_b/ρ_b. The sum is in theory performed over all the particles b of the domain. However, the use of kernel compact supports of radius h_t, proportional to the smoothing length h allows a reduction in the number of particles b which contribute to the sum in Eq. (6) and, thus to reduce the computational time. Consequently, only particles b located in the sphere of radius h_t and centered in a contribute to the evaluation of the function A relative to particle a. Equations (1)-(3) are integrated by SPH formalism and solved in the Spartacus-3D code [2] [3], developed at EDF R&D since 1999, mainly for coastal and environmental applications [4].

2. PARALLELISATION

Three main steps are followed in the parallel version of the code, during a time step:

- Step 1: Generation of a list of particles,
- Step 2: Search for particle links and link-based particle re-indexing,
- Step 3: Resolution of the equations.

The profiling of the SPH scalar version of the code shows that up to 60% of the CPU time is spent during Steps 1 and 2. The link-based particle re-indexing is important to reduce the time spent to calculate the operators.
SPH operators are expressed in terms of differences or sums of contribution of particles a and b, with a summation on b. Since spline kernels have a compact support, each particle a is only linked to its closest neighbours b for which $|\mathbf{r} - \mathbf{r}_b| < h_t$.
Most arrays are local and their size is computed from the number of particles located on a given processor augmented by the number of particles playing a role in the calculation of the operators. This allows to save memory and to avoid global communications, while building the operators. The hybrid OpenMP-MPI is implemented allowing up to 4 threads.

3. RESULTS

The dam breaking case is investigated to demonstrate the ability of Spartacus-3D to compute 3-D free surface flows. The experiment performed by the MAritime Research Institute Netherlands (MARIN) is described in [5]. As shown in Fig. 1, four vertical water elevation probes have been used: one in the reservoir and three in the tank.

3.1 Performance

The simulations are carried out on 1 rack of Daresbury Laboratory's BlueGene/P (1,024 nodes of quad-core 850MHz PowerPC), with MPI only and with the hybrid OpenMP-MPI approach. $1,000$ iterations of the dam-breaking case are performed.

Figure 2 shows that for a relatively small number of particles (1.3M, which means less than 320 particles per core for $4,096$ cores when the whole rack is used), the behaviour of the code is linear up to $2,048$ cores for all the simulations and the code only starts to lose efficiency going from $2,048$ to $4,096$ cores. This is due to the fact that the calculation part is not long enough to hide the communications between processors.

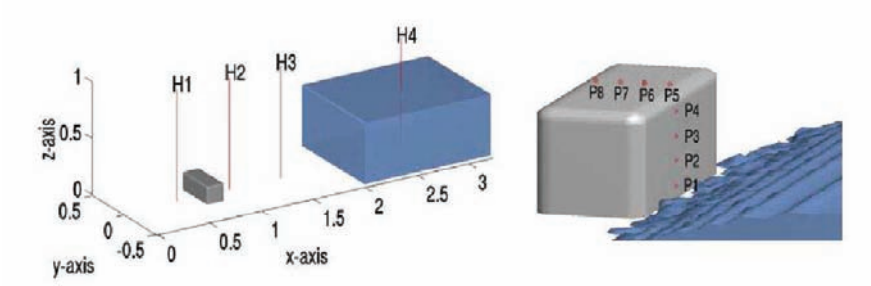


Fig. 1: *Geometry of the dambreak. Left: measurement positions for water elevation in the dambreak experiment [5]. Right: zoom of the box.*

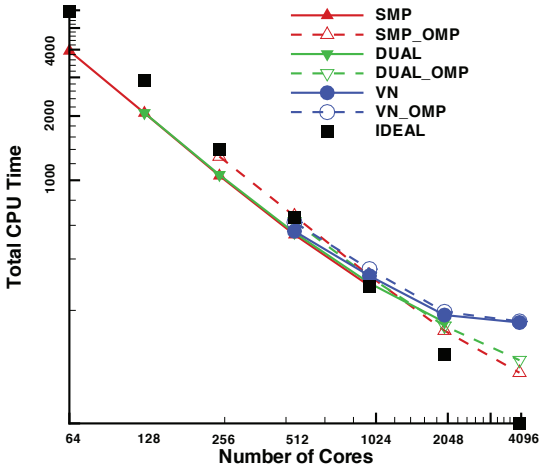


Fig. 2: *Total CPU time against number of cores for a* $1.3M$ *particle simulation.*

The fastest run is carried out in hybrid OpenMP-MPI mode, for $4,096$ cores, which confirms the efficiency of the hybrid OpenMP-MPI approach for multi-core machines.

4. COMPARISON WITH EXPERIMENT

The comparison with experiment is carried out on the water elevation in the four probes. Figure 3 shows that the water elevation computed by the simulation matches the experiment before the water impacts the obstacle. After this stage, a shift of phase that remains to be explored occurs, for all the probes.

REFERENCES

1. Monaghan, J.J. (1992). *Annual Review of Astronomy and Astrophysics*. 30, 543.
2. Moulinec, C., Issa, R., Marongiu, J.C. and Violeau, D. (2008). *Computer Modelling in Engineering and Science*. 25, 133.
3. Moulinec, C., Issa, R., Latino, D., Vezolle, P., Emerson, D.R., and Gu, X.J. (2008). High-Performance Computing and Smoothed Particle Hydrodynamics, Parallel CFD Conference.
4. Issa, R., Lee, E.S., Violeau, D., and Laurence, D.R. (2005). *International Journal for Numerical Methods in Fluids*. 47, 1101.
5. Kleefsman, K.M.T., Fekken, G., Veldman, A.E.P, Iwanowski, B., and Buchner., B. (2005). *Journal of Computational Physics*. 206, 363.

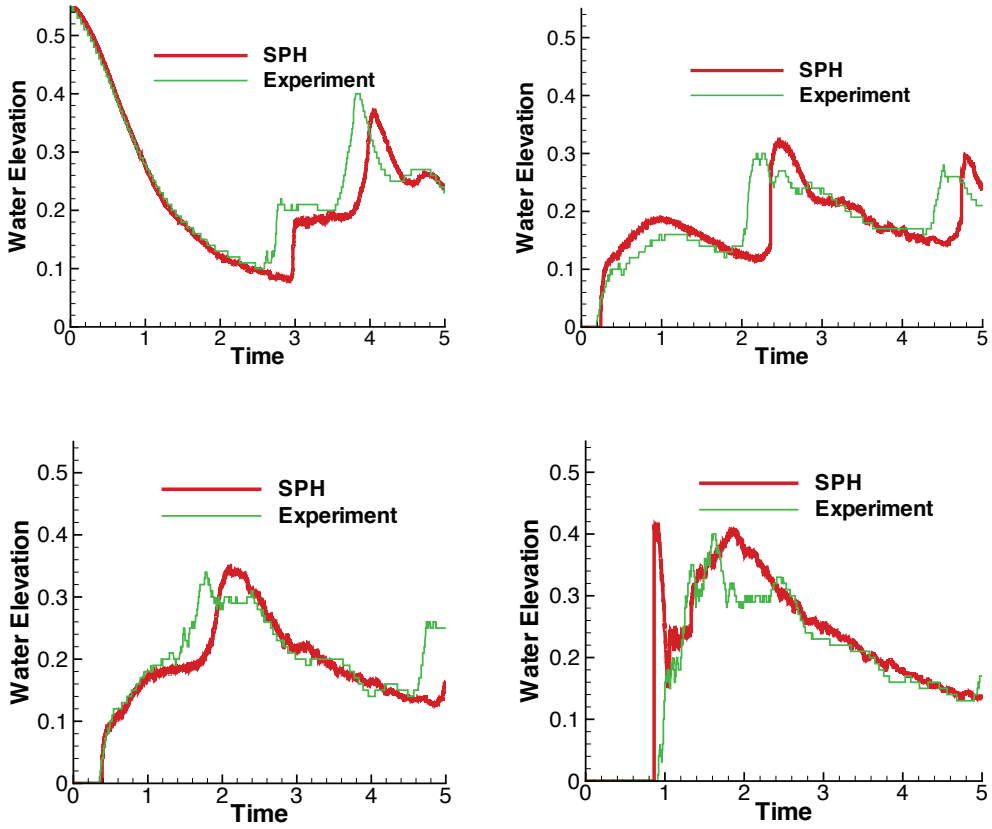


Fig. 3: *Comparison between the SPH* $1.3M$ *initial particle distribution and experiments. Water elevation in 4 probes.*

Parallel Implementation of Panel-free Boundary Conditions for the Vortex Particle Method

Felipe A. Cruz*, Christopher D. Cooper, Rio Yokota*, L. A. Barba***,**

*Dept of Mathematics, University of Bristol, Bristol, United Kingdom
**Universidad Técnica Federico Santa María, Valparaíso, Chile
***Dept of Mechanical Engineering, Boston University

Abstract: In this work, we simulate two-dimensional viscous incompressible flows around submerged bodies, based on a vortex particle method with re-meshing formulation. A panel-free approach is used for the treatment of the boundary conditions of the submerged bodies. The no-slip and no-through boundary conditions of a solid body immersed in a flow are fulfilled by placing a vortex sheet of a given strength γ at the walls of the body, and then diffusing the vortex sheet into the flow. For this, we use a panel-free method based on radial basis functions (RBF) to find the strength of the vortex sheet, and to diffuse the vortex sheet. The numerical implementation of the method makes use of a combination of fast parallel algorithms to speed-up the many stages of the simulations: parallel fast summation methods are used for the evaluation of the velocity and vorticity fields; and, an ad-hoc fast linear solver is implemented for the solution of dense linear systems of equations.

Keywords: meshfree, particle, parallel, vortex method, panel method

1 INTRODUCTION

In the viscous vortex particle method, the vorticity field is described by a Lagrangian representation. This is achieved by discretizing the vorticity field into N vortex particles which then move with the local velocity. The velocity field is represented in the following way:

$$\boldsymbol{\omega}(\mathbf{x},t) \approx \boldsymbol{\omega}_\sigma(\mathbf{x},t) = \sum_{i=1}^{N} \boldsymbol{\Gamma}_i(t)\zeta_\sigma\left(\mathbf{x}-\mathbf{x}_i(t)\right), \tag{1}$$

where Γ_i is the circulation of a particle, x_i corresponds to the position of a particle, and ζ_σ corresponds to the characteristic distribution of vorticity of each node. Often, ζ_σ is represented by a gaussian kernel. From the vorticity field obtained from equation (1), the velocity field can be recovered by means of the Biot-Savart law,

$$\mathbf{u}(\mathbf{x},t) = (\mathbf{K} * \omega)(\mathbf{x},t) = -\frac{1}{2\pi}\sum_{i=1}^{N} \boldsymbol{\Gamma}_i \mathbf{K}(\boldsymbol{x}_j - \boldsymbol{x}_i) + \boldsymbol{U}_0(\boldsymbol{x}_i,t) \tag{2}$$

where $\mathbf{K} = \nabla \times \mathbf{G}$, $\mathbf{G}$ corresponds to Green's function of the Poisson equation, and $\boldsymbol{U}_0$ is the free stream velocity. More details in the vortex method can be found in [11], [1].

As it can be seen from equation (1) and (2), in order to compute the value of vorticity and velocity for one particle location, the effect of all other $N-1$ particles must be considered. Therefore, if a naive approach is used to evaluate the velocity and vorticity fields at all N particle locations in the system, the total work will be in the order of $\mathcal{O}(N^2)$ calculations. Furthermore, real engineering problems, where certain acceptable accuracy is desired, may involve millions of particles. When using naive implementations for the evaluation of equations (1) and (2), simulations in the order of tens of thousands of particles become unpractical due to the limited computational resources that are available. However, in the following paper, a parallel version of the code is used, and methods that decrease the scheme to $\mathcal{O}(N)$, such as the Fast Multipole Method and Fast Gauss Transform are implemented.

1.1 Boundary conditions for the vortex particle methods

Formulating the boundary conditions on a solid wall is notoriously problematic in vortex methods. The difficulty arises due to the absence of a vorticity boundary condition for the Navier-Stokes equation, equivalent to no-through and no-

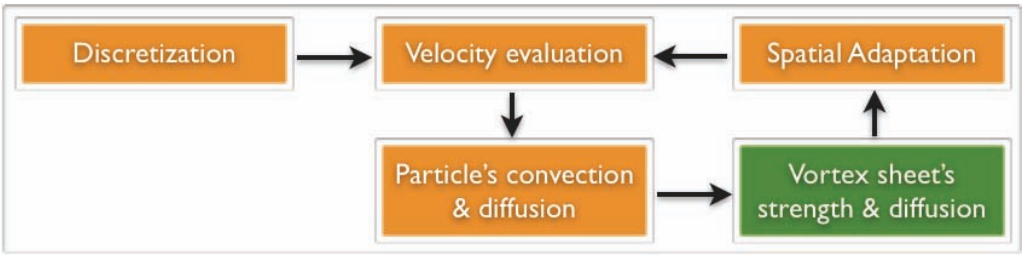


Figure 1. This diagram shows the vortex method algorithm when solving a problem of flow with a solid boundary. One loop in the diagram corresponds to one 'time step'. The orange colored boxes correspond to the steps of the vortex particle method without considering boundary conditions. The green colored box represents the step where the boundary conditions of the solid are considered.

slip at the wall. The problem has been addressed by means of a model of vorticity creation at the solid wall, initially proposed by Chorin [2]. The idea is to place a vortex sheet around the boundary, in order to nullify the spurious velocity generated there after the inviscid substep.

As described in [10], [9] and [4], the following equation to obtain the strength of the vortex sheet that produces a velocity discontinuity of $\mathbf{u}_{slip}$,

$$\gamma(s) - \frac{1}{\pi} \oint [\frac{\partial}{\partial n}[\log \mid \mathbf{x}(s) - \mathbf{x}(s') \mid] - \frac{\rho_1(s)}{L}]\gamma(s')\mathrm{d}s' = -2\,\mathbf{u}_{slip} \cdot \hat{s} \tag{3}$$

here, $\gamma(s)$ is the strength of the vortex sheet, $\mathbf{u}_{slip}$ is the velocity discontinuity created by the vortex sheet, $\hat{s}$ is a unitary vector tangent to the solid in the direction of integration, ρ_1 is the eigenfunction of the first term in the spectral decomposition of the kernel, and L the perimeter of the body.

Placing the vortex sheet found with (3), around the boundary, the spurious slip velocity generated in the inviscid step is cancelled. After obtaining the vortex sheet, this needs to be transfered to the nearby particles in the fluid domain. This is done by solving a diffusion equation with the correct boundary conditions (a, b, c).

$$\frac{\partial \boldsymbol{\omega}}{\partial t} - \nu \Delta \boldsymbol{\omega} = 0, \qquad \boldsymbol{\omega}(t - \delta t) = 0, \qquad \nu \frac{\partial \boldsymbol{\omega}}{\partial n} = \frac{-\gamma(s)}{\delta t} \tag{a, b, c}$$

1.2 Algorithm description

The complete algorithm for the vortex particle method with boundary conditions can be summarized in the following stages: discretization of the flow into particles, velocity evaluation at the particle's location, convection and diffusion of the particles, solving vortex sheet's strength and diffusion, and spatial adaptation. A diagram that represents the different stages of the algorithms is presented in Figure 1. Specific details of the stages of the algorithms can be found in [1], [3].

2 PANEL-FREE BOUNDARY CONDITIONS FOR THE VORTEX PARTICLE METHOD

2.1 Radial basis functions

Radial basis function method (RBF) is often used to interpolate an unknown function. This is done by representing the function as a linear superposition of basis functions:

$$f(\mathbf{x}) \approx \sum_{i=1}^{N} \phi(\mid \mathbf{x} - \mathbf{c}_i \mid)\alpha_i \tag{4}$$

where the function $f(x)$ is represented as a linear combination of basis functions (ϕ), each centered at a point c_i and with weight α_i. Then, for a given set of basis functions, the idea is to choose the coefficients α_i such that equation (4) is satisfied at the collocation points. As it was shown by Kansa in [7], the RBF function approximation can be used as a method in order to solve partial differential equations and also integral equations.

In this work, we use an RBF approach to enforce the no-slip and no-through boundary conditions of a solid body immersed. For this, we solve the equations presented in §1.1, we make use a panel-free method based on radial basis functions (RBF) to find the strength of the vortex sheet, and to diffuse the vortex sheet.

2.2 Finding the vortex sheet strength

The first step is to discretize the surface of the solid into a set of points. The points will constitute the RBF centers for the computation of the vortex sheet strength, γ. Then, the slip velocity, which was caused by the previous inviscid sub-time step, is computed using equation (2) at the surface of the boundary.

Having the slip velocity, it is possible to obtain γ from (3). The value of ρ_1 can also be solved with an RBF approach, using the same principles that will be used when solving the Fredholm equation of the second kind, which will be described in detail below. For more details on ρ_1, see [8]. Let the RBF approximation of the function of $\gamma(s)$ around the surface of the solid body be

$$\gamma(\mathbf{x}) \approx \sum_{i=1}^{N} \phi(\mid \mathbf{x} - \mathbf{c_i} \mid)\alpha_i \tag{5}$$

Now, applying (5) to equation (3) at every boundary node position, we get the following system of equations

$$[\phi_{ki} - \Theta_{ki} + \Lambda_{ki}] \cdot [\alpha_i] = [u^i_{slip} \cdot s_i] \tag{6}$$

where $\mathbf{x} = \mathbf{x}_k$ and

$$\Theta = \oint \frac{1}{\pi}\frac{\partial}{\partial n}[\log \mid \mathbf{x} - \mathbf{x}' \mid]\phi(\mid \mathbf{x}' - \mathbf{c}_i)\mathrm{d}\mathbf{x}', \qquad \Lambda = \oint \frac{\rho_1}{L}\phi(\mid \mathbf{x}' - \mathbf{c}_i \mid)\mathrm{d}\mathbf{x}'$$

Solving this linear system of equations, we will find the coefficients of the RBF representation that best fits the function of the strength of the vortex sheet around the surface of the body.

2.3 Diffusion of the vortex sheet

After the strength of the vortex sheet on the body is found, such that it cancels the slip velocity, it is necessary to diffuse the vorticity that was created on the surface to the surrounding fluid. This step is based on the solution of the PDE problem described in (a, b, c), once again, with an RBF approach, as described in [12]. An RBF approximation of the vorticity for the whole field can be written as,

$$\omega^t(x) = \sum_{i=1}^{N} \beta_i^t \phi(\mid \mathbf{x} - \mathbf{c}_i \mid) \tag{7}$$

By using the RBF representation for the PDE, and the boundary condition (knowing that $\frac{\partial \omega}{\partial n} = \nabla\omega \cdot n$), treating the time differential operator with a Crank-Nicholson scheme, and applying this to every node in the domain, we can write the following matrix equation:

$$[\phi - \delta t\nu\theta\Delta\phi][\beta^t] = [\phi + \delta t\nu(1-\theta)\Delta\phi][\beta^{t-\delta t}] \tag{8}$$

for nodes that are located on the boundaries, the Neumann boundary condition shown in (a, b, c) has to be enforced:

$$\sum_{i=0}^{N} \beta_i^t \nabla\phi(\mid \mathbf{x} - \mathbf{c}_i \mid) \cdot n = \frac{-\gamma^t(s)}{\nu\delta t} \tag{9}$$

and by solving for the domain and boundary nodes the following equation is obtained:

$$\begin{bmatrix} A \\ B \end{bmatrix} \cdot [\beta] = \begin{bmatrix} C \\ 0 \end{bmatrix} \tag{10}$$

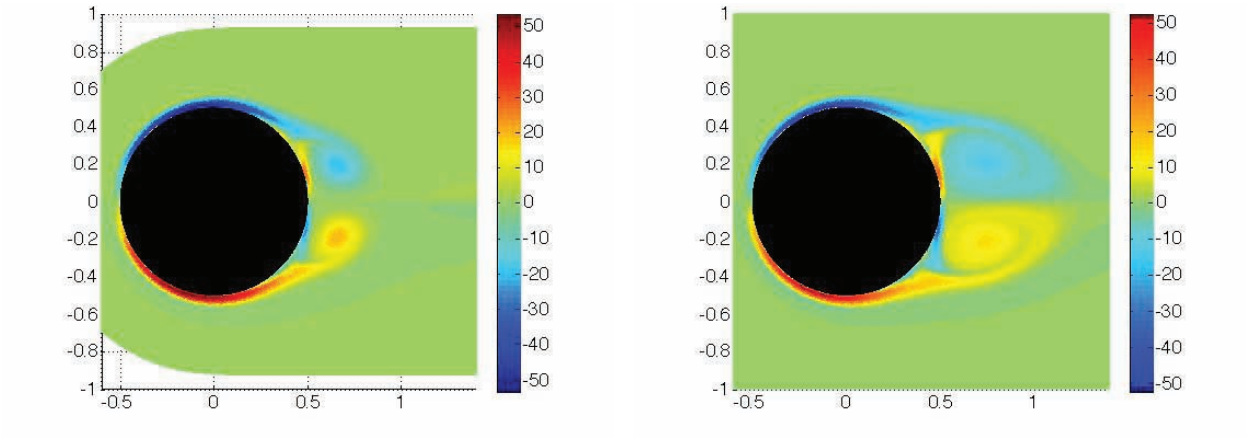


Figure 2. Preliminary results on the flow around a circular cylinder. These figures show the vorticity generated around the cylinder.

where A is an $N \times M$ matrix with $\nabla\phi_{ij} \cdot n_i$ as coefficients; B is a $M-N \times M$ matrix whose values are $\phi_{ij} - \delta t \nu \theta \Delta \phi_{ij}$, and i runs from $N+1$ to M. β is a vector with the coefficients of the RBF approximation shown in (7). C is a vector of size N, with $\frac{\gamma_i}{\nu \delta t}$ as values and finally, the remaining $M-N$ positions of the RHS are 0.

Here we have M nodes in the whole domain (domain and boundary), from which the first N nodes are located on the boundary ($M-N$ is the number of nearby particles considered for diffusion). Solving this system, the coefficients of the RBF expansion in (7) are obtained. Then, using (7), the vorticity field diffused from the vortex sheet is computed.

The position of the domain nodes for this RBF scheme are the positions of the existing vortex particles. However, the core size of the basis function, which is a Gaussian, is not necessarily the same as the core size of the particles. This is an advantage, since we can fix the value of the basis function's core size as we wish, being a very important factor in the accuracy of any RBF method.

3 NUMERICAL METHOD AND PRELIMINARY RESULTS

The implementation of the method makes use of several components in order to speed-up the computations. The focus is centered around the computational bottlenecks of the algorithm, namely: the velocity evaluation, the vorticity evaluation, and the solution of a dense system of equations. In order to obtain results within a reasonable computational time, these computationally intensive steps are efficiently computed by the following algorithms: parallel fast multipole method [5], fast gauss transform [6], and a fast parallel solver for dense systems of equations. Parallel efficient implementations of the algorithms are currently being integrated into the main simulation code, and numerical results using the fully parallel version of the code are going to be presented at the conference.

Using a not fully optimized implementation, we present some preliminary results. As a test, we have computed the flow around a circular cylinder. Our approach has been tested for an impulsively started flow, for Reynolds number Re = 200. Figures 2(a) and 2(b) show the vorticity field at different time steps of the calculation. The number of particles in the system is in the order of tens of thousands of vortex particles.

4 CONCLUSIONS

In this work, we simulate two-dimensional viscous incompressible flows around submerged bodies, based on a vortex particle method with re-meshing formulation. A panel-free approach is used for the treatment of the boundary conditions of the submerged bodies. Parallel optimized versions of the fast multipole method, fast gaussian summation, and fast solution of dense system of equations are currently being integrated into the simulation code. For the conference date, we will present a fully parallel implementation, which would allow us to try much more interesting problems, such as multi-body simulations.

REFERENCES

[1] L. A. Barba. *Vortex method for computing high-Reynolds number flows: Increased accuracy with a fully mesh-less formulation*. PhD thesis, California Institute of Technology, 2004.

[2] A. J. Chorin. Numerical study of slightly viscous flow. *J. Fluid Mech.*, 57:785–796, 1973.

[3] C. D. Cooper and L. A. Barba. Panel-free boundary conditions for viscous vortex methods. In ., 2009.

[4] G.-H. Cottet and P. Koumoutsakos. *Vortex Methods. Theory and Practice*. Cambridge University Press, 2000.

[5] Felipe A. Cruz, L. A. Barba, and Matthew G. Knepley. Fast multipole method for particle interactions: an open source parallel library component. 20th Parallel Computational Fluid Dynamics Conference, Lyon, May 2008.

[6] L. Greengard and J. Strain. The fast Gauss transform. *SIAM J. Sci. Statist. Comput.*, 1991.

[7] E. J. Kansa. Multiquadrics —A scattered data approximation scheme with applications to computational fluid-dynamics, II. Solutions to parabolic, hyperbolic and elliptic partial differential equations. *Computers Math. Applic.*, 19(8/9):147–161, 1990.

[8] P. Koumoutsakos and A. Leonard. Improved boundary integral method for inviscid boundary condition applications. *AIAA J.*, 31(2):401–404, 1993.

[9] P. Koumoutsakos and A. Leonard. High-resolution simulations of the flow around an impulsively started cylinder using vortex methods. *J. Fluid Mech.*, 296:1–38, 1995.

[10] P. Koumoutsakos, A. Leonard, and F. Pépin. Boundary conditions for viscous vortex methods. *J. Comp. Phys.*, 113:52–61, 1994.

[11] A. Leonard. Vortex methods for flow simulation. *J. Comp. Phys.*, 37:289–335, 1980.

[12] M. Zerroukat, H. Power, and C. S. Chen. A numerical method for heat transfer problems using collocation and radial basis functions. *Int. J. Numer. Meth. Engng.*, 42(7):1263 – 1278, August 1998.

DNS of Homogeneous Turbulence Using Vortex Methods Accelerated by the FMM on a Cluster of GPUs

Rio Yokota *, Tetsu Narumi **, Ryuji Sakamaki **, Shun Kameoka **, Kenji Yasuoka **, Shinnosuke Obi **

**Dept of Mathematics, University of Bristol, Bristol, United Kingdom*
***Department of Mechanical Engineering, Keio University, Yokohama, Japan*

Abstract: Recent advances in the parallelizability of fast N-body algorithms, and the programmability of graphics processing units (GPUs) have opened a new path for particle based simulations. For the simulation of turbulence, vortex methods have been considered as an interesting alternative to finite difference and spectral methods. The present study focuses on the efficient implementation of the fast multipole method and pseudo-particle method on a cluster of NVIDIA GeForce 8800 GT GPUs, and applies this to a vortex method calculation of homogeneous isotropic turbulence. The results of the present vortex method agree quantitatively with that of the reference calculation using a spectral method. *Keywords:* Vortex Methods; Fast Multipole Methods; GPU clusters; Homogeneous Turbulence

1 INTRODUCTION

Particle-based simulations are a natural method for solving discrete systems such as in astrophysics, and molecular dynamics (MD). They also provide an interesting alternative to grid-based methods for solving continuum systems, as seen in smooth particle hydrodynamics (SPH) and vortex methods (VM). The computational cost of treating the continuum system as an N-body problem is relatively large compared to conventional grid based methods. However, recent changes in the hardware architecture are likely to offer a significant advantage to N-body problems.

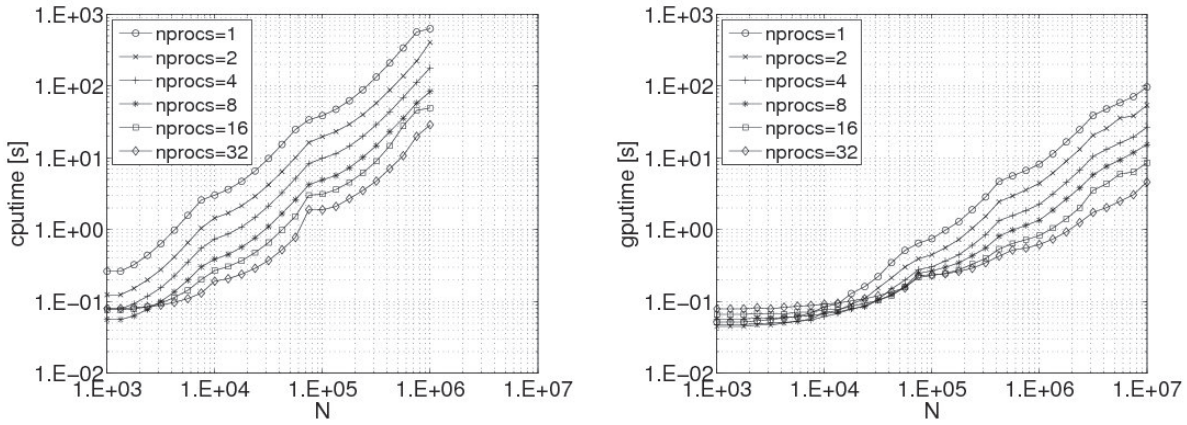
The traditional approach to accelerate N-body problems has been the use of hierarchical algorithms, such as the Barnes & Hut treecode [1] and the fast multipole method (FMM) [2]. A more recent trend in the field of N-body problems has been the efficient implementation of such hierarchical algorithms on GPUs. Stock & Gharakhani [3] implemented the treecode on the GPU to accelerate their vortex method calculation. Similarly, Gumerov & Duraiswami [4] calculated the Coulomb interaction using the FMM on GPUs. The relative acceleration ratio between the fast algorithm and direct calculation was $17/127 \approx 0.134$ for Stock & Gharakhani and $72/855 \approx 0.084$ for Gumerov & Duraiswami. The difference between the relative acceleration ratio is possibly caused by the difference in the data-parallelism of the treecode and FMM.

In the present study, we apply the fast multipole method on a cluster of 32 GPUs to the calculation of a homogeneous isotropic turbulence using vortex methods. The results are compared with a spectral method code to evaluate the relative performance of vortex methods when they are accelerated by GPUs.

2 CALCULATION METHOD

2.1 Vortex Method

The present vortex method uses Gaussian blobs as calculation elements. The vorticity equation is solved in a fractional step manner by solving for the convection, stretching, and diffusion separately. The velocity and stretching calculation are accelerated by the FMM and executed on the GPU. All other components are executed on the CPU. A convergent core spreading method is used to account for the diffusion. In this method the core radius of the vortex elements are reinitialized every 10 time steps and the vortex strength is calculated by radial basis function interpolation. The BICGSTAB method is used to solve the system of equations, while the FMM neighbor list is used to accelerate the calculation of vorticity.



(a) CPU (b) GPU

Figure 1. Calculation time of FMM on parallel CPUs and GPUs

2.2 FMM on GPUs

In the present calculations, we use NVIDIA's GeForce 8800GT, which has 112 streaming processors where 8 streaming processors are grouped into a multiprocessor. Each multiprocessor has 16 Kbyte of shared memory, which can be accessed at high speed. The clock frequency of the multiprocessor of our XFX's GPU is 1,562 MHz, where 2 single precision floating point operations can be performed per clock cycle. Thus, the peak performance of the GeForce 8800 GT is 350 Gflops ($= 112 \times 1,562 \times 2$).

In order to execute the FMM efficiently on this architecture, the following modifications were made to the FMM. First, the complex spherical harmonics were transformed to real basis functions, in order to avoid complex arithmetic on the GPU. Second, in order to minimized the memory usage per interaction, all of the tranlation matrices were generated on-the-fly. Third, the box structure and interaction list of the FMM are restructured and renumbered to match the number of threads per block on the GPU, so that no threads remain idle.

2.3 Parallelization

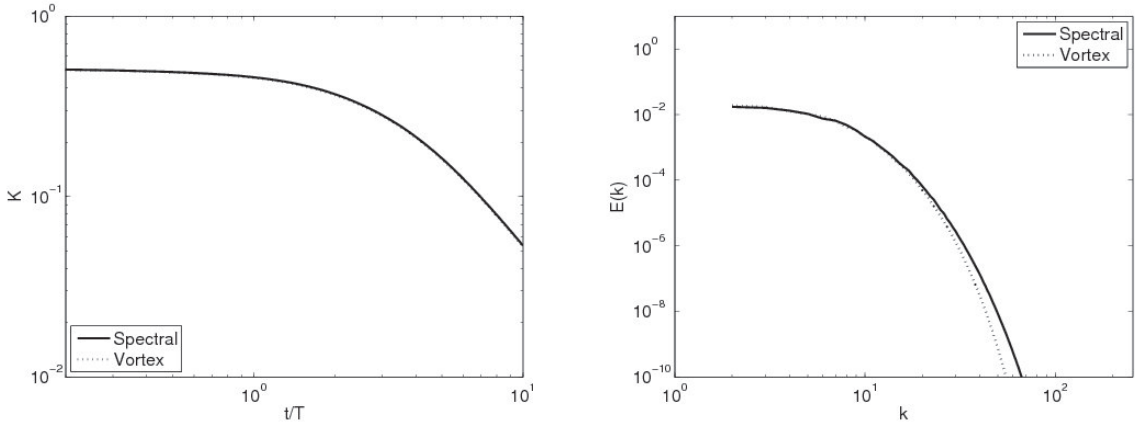
We used 32 nodes of PCs (HP xw4600), each of them having a dual-core CPU (Intel Core2 Duo E6850), 4Gbyte of memory, and a GPU (XFX PV-T88P-YDQ4 which has NVIDIA's GeForce 8800 GT). The GPUs were plugged in via 16x PCI Express 2.0 slots. The PCs were connected by a gigabit ethernet network through a 48-port HUB (NETGEAR GS748TS-100JPS).

The present study involves the parallelization of the FMM on distributed memory architectures using MPI. Balancing the computational workload and amount of data transfer between the processes is a challenging task for adaptive FMMs. However, for the present calculation of the homogeneous turbulence, the particle density remains constant throughout the entire domain and varies little over time. Furthermore, the periodic boundary condition prevents the load of the boarder cells from becoming small. Therefore, the workload between different processes can be balanced by simply partitioning the parallel computation domain according to the oct-tree structure of the FMM.

3 CALCULATION RESULTS

3.1 FMM on a cluster of GPUs

In order to evaluate the performance of the parallel GPU calculation, we measured the performance of the velocity calculation for both parallel CPUs and parallel GPUs. The particles were randomly positioned in a $[-\pi, \pi]^3$ domain, and given a random vortex strength between 0 and $1/N$. The core radius was set to $\sigma = 2\pi N^{-1/3}$, which results in an average overlap of $\sigma/\Delta x = 1$. The CPU used here is a Intel Core2 Duo E6850 and the code is written in Fortran 90 and compiled on a Intel Fortran compiler 10.0 with the "-openmp" option. The GPU is a NVIDIA GeForce 8800 GT and the Fortran code calls a CUDA library, which was compiled with NVCC using the "-use_fast_math" option.



(a) Decay of kinetic energy

(b) Energy spectra at $t/T = 10$

Figure 2. Decay of kinetic energy and energy spectra of the homogeneous turbulence calculation

Figure 1(a) shows the calculation time of the FMM on parallel CPUs, while Figure 1(b) shows the calculation time on parallel GPUs, where N is the number of calculation points and $nprocs$ is the number of processes. Even though the calculations on the CPU were performed until $N = 10^6$, the range of the axes is kept the same for both plots. By comparing the results for $N = 10^6$ between the two plots, it can be seen that the GPU is approximately two orders of magnitude faster. However, the scalability of the parallel GPU calculation is quite small for $N < 10^5$, while the scalability of the parallel CPU calculation remains large even for small N. This is caused by the inability to exploit the full computational power of GPUs when the number of calculation points per process is small ($N < 10^4$). Another cause may be that our code has inefficient parts that constantly take around $0.1s$ regardless of N, $nprocs$, or the whether it is processed on a CPU or GPU. Thus, the scaling appears to be bad for any run that should theoretically take less than $0.1s$. The justification for having such routines in our code is based on the fact that we would rather have a generalized FMM that can handle different equations and different hardware architecture at the cost of adding $0.1s$ to our total execution time, than to limit the capability of our code just to improve the scalability for small problems.

It maybe worth noting that the equation calculated here is the velocity calculation for vortex methods with elliptic Gaussian smoothing functions, which is approximately an order of magnitude more time consuming than a simple potential calculation. We have indeed confirmed that our code can calculate the potential function for $N = 10^6$ in less than $1s$ on a single GPU.

4 CALCULATION OF ISOTROPIC TURBULENCE

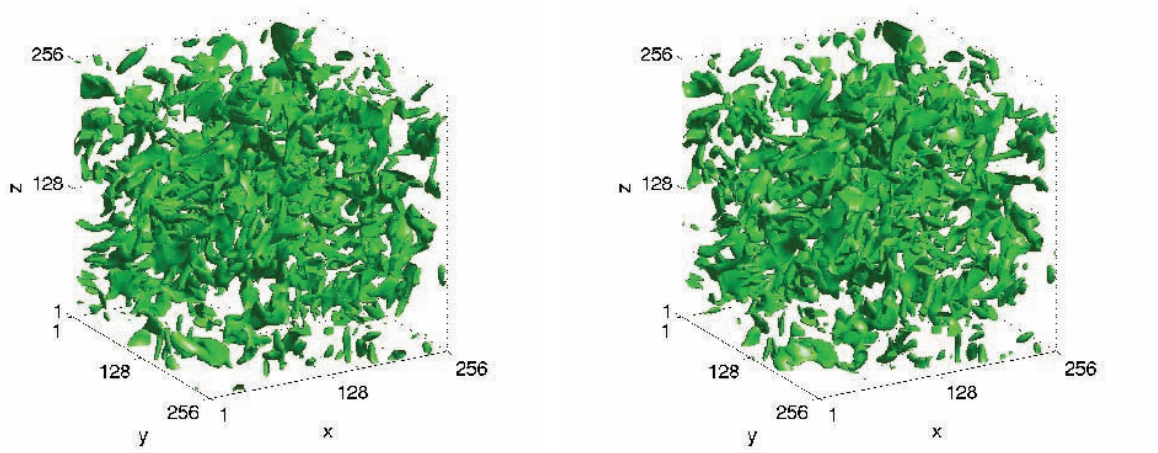
4.1 Calculation Conditions

The flow field of interest is a decaying isotropic turbulence with an initial Reynolds number of $Re_\lambda \approx 100$. The calculation domain is $[-\pi, \pi]$ and has periodic boundary conditions in all directions. In the present vortex method calculation, the periodic boundary condition is approximated by the use of periodic images. Details of the periodic FMM are shown in our previous publication [8]. The order of multipole expansion was set to $p = 10$, and the number of periodic images was $2^5 \times 2^5 \times 2^5$ for the present calculations. We used a total of 32 GPUs for the calculation of the isotropic turbulence.

The spectral Galerkin method with primitive variable formulation is used in the present study as reference. A pseudo-spectral method was used to compute the convolution sums, and the aliasing error was removed by the 3/2-rule. The time integration was performed using the fourth order Runge-Kutta method for all terms. No forcing was applied to the calculation, since it would be difficult to do so with vortex methods.

4.2 Calculation Results

Figure 2(a) shows the decay of kinetic energy. $Spectral$ is the spectral method and $Vortex$ is the vortex method calculation. The time is normalized by the eddy turnover time T. The homogeneous isotropic turbulence does not have



(a) Spectral method (b) Vortex method

Figure 3. Isosurface of the second invariant of the velcoity derivative tensor

any production of turbulence, and thus the kinetic energy decays monotonically with time. This decay rate is known to show a self-similar behavior at the finial period of decay. This is confirmed by the straight drop of K that appears at the end of this log-log plot. The results of the two methods agree perfectly until $t/T = 10$, where the kinetic energy drops an order of magnitude from the initial value.

Figure 2(b) shows the energy spectrum at $t/T = 10$. k is the wave number, and E(k) is the kinetic energy contained in the wave number k. The Reynolds number is $Re_\lambda \approx 50$, and at this Reynolds number it is difficult to observe an inertial subrange of $k^{-5/3}$, nor a k^4 behavior at low wave numbers. The results of the two methods are in good agreement, except for the fact that the vortex method slightly underestimates the energy at higher wave numbers.

The isosurface of the second invariant of the velocity derivative tensor $II = u_{i,j}u_{j,i}$ at time $t/T = 10$ is shown in Figure 3. The result of the spectral method and vortex method are almost indistinguishable. These results indicate the soundness of the present vortex method calculation using GPUs. It is fair to say that the single precision calculation of the velocity does not have any detrimental effect on the final accuracy of our turbulence simulations. Furthermore, the calculation of the present vortex method on 32 GPUs took nearly the same amount of time as the spectral method on 32 CPUs.

5 CONCLUSION

The calculation using the GPU (GeForce 8800GT with nvcc -use_fast_math) is approximately 100 times faster than the CPU (Intel Core2 Duo E6850 with ifort -openmp) calculation. The parallelization efficiency of the FMM on parallel GPUs is relatively low when the number of calculation points is small ($N < 10^5$), whereas the parallel CPU scaling is good even for $N < 10^4$. However, the scaling of the parallel GPU implementation is good for problems with larger N, and the velocity calculation for $N = 10^7$ vortex elements took approximately $4.6s$.

The present acceleration technique enabled the calculation of a homogeneous isotropic turbulence using a relatively large number of vortex elements. The kinetic energy decay and energy spectrum of the well resolved vortex method calculation agreed quantitatively with that of the reference calculation using a spectral method. Such accuracy for completely meshless turbulence simulations have not been reported previously. Furthermore, the calculation of the present vortex method on 32 GPUs took nearly the same amount of time as the spectral method on 32 CPUs.

ACKNOWLEDGEMENT

This study was partially supported by the Core Research for Evolution Science and Technology (CREST) of the Japan Science and Technology Corporation (JST).

REFERENCES

[1] J. E. Barnes, P. Hut, Nature 324 (1986) 446.
[2] L. Greengard, V. Rokhlin, J. Comput. Phys. 73 (1987) 325.
[3] M. J. Stock, A. Gharakhani, in: Proc. of 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada (2008).
[4] N. A. Gumerov, R. Duraiswami, J. Comput. Phys. 227 (2008) 8290.
[5] M. Yokokawa, K. Itakura, A. Uno, T. Ishihara, Y. Kaneda, in: Proc. of SC2002, Baltimore, Maryland (2002)
[6] R. T. Fisher, L. P. Kadanoff, D. Q. Lamb, A. Dubey, T. Plewa, A. Calder, F. Cattaneo, P. Constantin, I. Foster, M. E. Papka, S. I. Abarzhi, S. M. Asida, P. M. Rich, C. C. Glendenin, K. Antypas, D. J. Sheeler, L. B. Reid, B. Gallagher, S. G. Needham, IBM J. Res. & Dev. 52 (2008) 127.
[7] G.-H. Cottet, B. Michaux, S. Ossia, G. VanderLinden, J. Comput. Phys. 175 (2002) 702.
[8] R. Yokota, T. K. Sheel, S. Obi, J. Comput. Phys. 226 (2007) 1589.

MECHANICAL/AEROSPACE ENGINEERING APPLICATIONS

Optimization of Synthetic Jet Parameters over an Elliptical Profile Using Response Surface Methodology

Engin Erler*, Ismail H. Tuncer, Myong H. Sohn*****

**Middle East Technical University, Dept. of Aerospace Engineering, Ankara 06531, Turkey*
(e-mail: eerler@ae.metu.edu.tr)
*** Middle East Technical University, Dept. of Aerospace Engineering, Ankara 06531, Turkey*
(e-mail: tuncer@ae.metu.edu.tr)
**** Korea Air Force Academy, Dept. of Aerospace Engineering, Chungbuk-Do 363-849 Republic of Korea*
(e-mail: myongsohn@hanmail.net)

Abstract: Low speed flows over an elliptic profile in the presence of a synthetic jet and the Coanda effect are investigated numerically. The synthetic jet parameters including its location over the profile are optimized for minimum drag using a Response Surface Methodology. The flow solutions required by the design of experiment are computed in parallel in a PC cluster.

Keywords: Coanda effect, synthetic jet , design of experiment, active flow control

1. INTRODUCTION

In recent years, the studies show that, high lift/low drag requirement of an aircraft may be achieved by active flow control. Active flow control is the ability of controlling the flow with addition of energy and without attachment of auxiliary devices such as riblets, flaps or large-eddy breakup devices [1, 2]. Lift and thrust enhancement, drag reduction, noise abatement, stall delaying, full/partial flow reattachment, mixing enhancement are the main outcomes of active flow control.

One way of active flow control is the use of blowing jets from aerodynamic surfaces used in many applications for separation and circulation control. Blowing jets are mostly used for circulation control of the blunt trailing edges. Blowing air near the trailing edge tangential to the surface, attaches the separated flow to the circular surface. This tendency of a stream of fluid to stay attached to a convex surface, rather than follow a straight line in its original direction is called Coanda Effect, and an example is shown in Fig. 1.

Another way of active flow control is by using synthetic jets. A synthetic jet results from an oscillating membrane in an enclosed area with an orifice at the top. The oscillating membrane sucks the air from the flow, and blows the air back into the flow. In this method, the flow gains momentum with introducing zero-net mass flux into the flow.

The aim of this study is to optimize synthetic jet parameters over a 12.5% thick elliptic profile with and without using the Coanda effect. In preliminary studies, the numerical method is first validated with the presence of a steady-blowing jet. The optimization is carried out, next, without using the Coanda effect at zero angle of attack. The synthetic jet parameters are optimized to minimize the drag coefficient while keeping the jet power constant. In full paper the synthetic jet parameters for the Coanda effect, will also be optimized for the maximum downstream location of the separation point at both low and high angle of attacks.

2. NUMERICAL METHOD

Unsteady, turbulent flows over a 12.5% thick elliptic profile are computed by a Navier-Stokes solver. A jet is implemented by imposing the velocity boundary conditions on the wall. It is defined by six parameters; the jet type, the jet velocity (u_{jet}), the jet frequency (F_{jet}), the jet location (x_{jet}), the jet slot width (w_{jet}) and the jet angle (α_{jet}). The jet parameters are then optimized for minimum drag using Response Surface Methodology (RSM). In RSM, the response surface for the objective function is represented by a 2nd order polynomial function:

$$f(u_{jet}, x_{jet}, \alpha_{jet}, F_{jet}) = C_1 u_{jet}^{\ 2} + C_2 x_{jet}^{\ 2} + C_3 \alpha_{jet}^{\ 2} + C_4 F_{jet}^{\ 2} + C_5 F_{jet} u_{jet} + C_6 u_{jet} \alpha_{jet} + C_7 u_{jet} x_{jet} + C_8 \alpha_{jet} F_{jet} + C_9 F_{jet} x_{jet} + C_{10} \alpha_{jet} x_{jet} + C_{11} u_{jet} + C_{12} x_{jet} + C_{13} \alpha_{jet} + C_{14} F_{jet} + C_{15}$$

The unknown coefficients in the response function are evaluated by Least Squares method following a certain number of function evaluations based on a design of experiment. Box-Behnken and Full Factorial methods may be used for the design of experiment. The corresponding function evaluations needed are given in table 1:

Table 1: Number of flow solutions needed for Box-Behnken and Full Factorial methods

Number of Optimization Variables	Number of Flow Solutions for Box-Behnken Method	Number of Flow Solutions for Full Factorial Method
3	15	27
4	27	81
5	46	243

2.1 Parallel Computations

The RSM method lends itself to parallel computations inherently. The function evaluations, which, in this case, are unsteady flow solutions, may all be performed in parallel. In additions, each flow solution is computed in parallel based on domain decomposition. PVM library routines are used for Interprocess communication in the parallel computations. Computations are performed in a 64 processor Linux cluster.

4. PRELIMINARY RESULTS

In an earlier study, the flow solutions with and without a steady jet over a 12.5% thick elliptical profile are first validated [3-6].The validation of the Coanda effect is also given in Figures 1 and 2. As shown, the flow field prediction and the distribution surface pressure are, in general, in agreement with the experimental and numerical studies of Shrewbury et al [2]

In a preliminary optimization study for minimizing the drag at an angle of attack of 0 degrees, 0.1 free stream Mach number and a Reynolds number of $1x10^{6,}$ the drag coefficient was reduced by 9.5%. The parallel computations took about 16 hours on 6 Intel quad-core processors (on 24 cores) with 2.33GHz clock speed.

REFERENCES

1. Glezer, A. and Amitay,M., "Synthetic Jets," Annual Review of Fluid Mechanics, Vol. 34, No. 1, 2002, pp. 503–529.
2. Shrewsbury, G., "Numerical study of a research circulation control airfoil using Navier- Stokes methods," Journal of Aircraft, Vol. 26, No. 1, 1989, pp. 29–34.
3. Erler, E. , Active Flow Control Studies Over an Elliptical Profile, M.S. Thesis, METU, 2008
4. Akçayöz, E., Numerical Investigation of Flow Control Over an Airfoil with Synthetic Jets and Its Optimization, M.S. Thesis, METU, 2008
5. Kaya, M., Path Optimization of Flapping Airfoils Based on Unsteady Viscous Flow Solutions, Ph.D. Thesis, METU, 2008
6. Akçayöz, E., Erler, E. and Tuncer, I. H., Flow Control Studies Over an Airfoil and an Elliptic Profile, 4th Ankara International Aerospace Conference, METU, Ankara, AIAC-2007-120, 10-12 September 2007

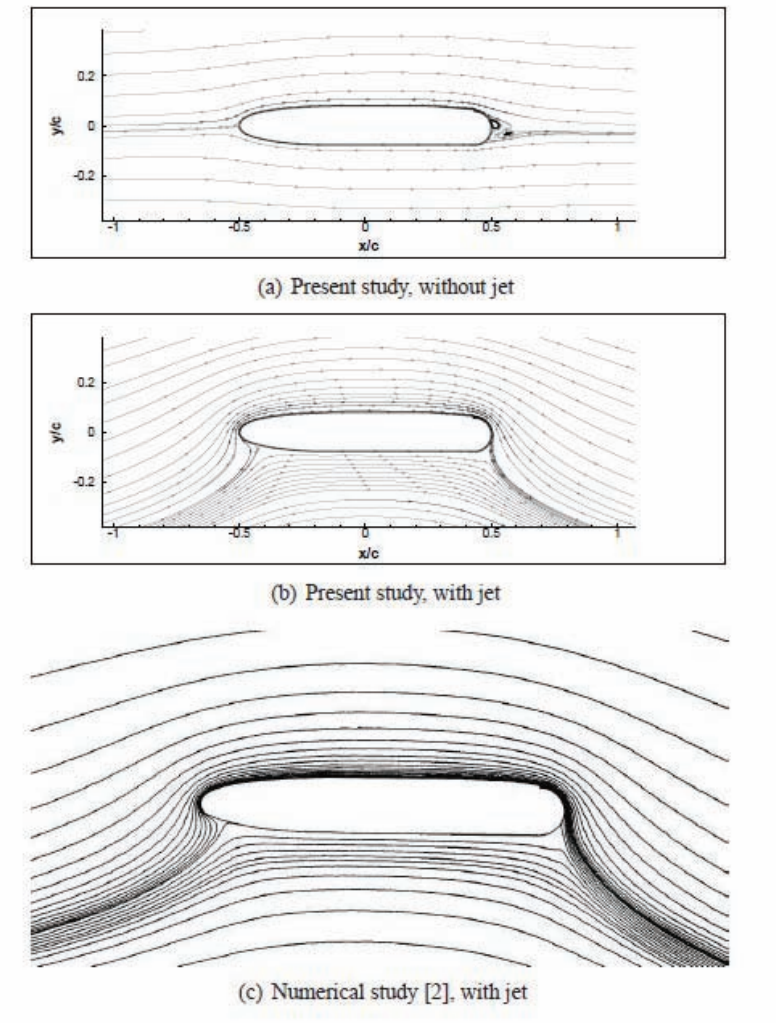


Fig. 1: The Flow around an elliptical profile [3]

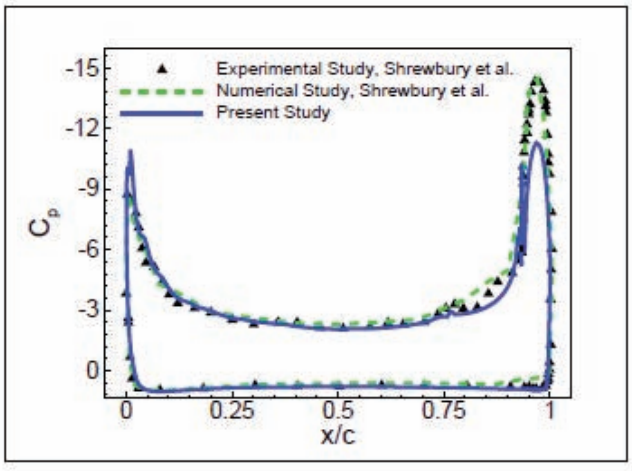


Fig. 2: Surface pressure distribution under the Coanda effect

Parallel Computations on Three Dimensional Aero-Acoustic Field past a Circular Cylinder

Tae soo Kim *, Jae soo Kim, Pa ul Mun***

** Department of Aerospace Engineering, Chosun University*
375 Seosuk-dong, Dong-gu, Gwang-ju, 501-759, Republic of Korea
*** corresponding author (Tel:82-62-230-7080; e-mail:jsckim@chosun.ac.kr)*

Abstract: The unsteady three dimensional compressible Reynolds averaged Navier Stokes equation was numerically computed to analyze the flow and aero acoustic fields past a circular cylinder by applying a high order and high resolution numerical method. While the numerical computation domain was set up wider than the methods with non-reflective characteristic boundary condition, the very simple boundary conditions were used. For the numerical result, the periodic characteristic of Strouhal Number due to vortex shedding was comparatively analyzed with other experiment values and two dimensional numerical results. Three dimensional Karman vortex shedding characteristic disturbed by secondary vortices and the propagation characteristic of pressure waves and acoustic waves were analyzed.

Keywords: Three Dimensional aero acoustic fields, Optimized High-Order Compact Scheme, Vortex Shedding, OpenMP Parallelized Computation, Secondary Vortex

1. INTRODUCTION

Even though research on the flow over a circular cylinder has been actively carried out up to the present, there are still many un-clarified phenomena such as wakes, secondary vortices, aerodynamic acoustic characteristics etc...[1].

The flow is symmetric with respect to the front and back if the Reynolds Number is less than 45. If the Reynolds Number is from 45 to 190, two dimensional vortex shedding will occur in the rear of the cylinder. If the Reynolds number is over 190, the vortex shedding changes to three dimensional unstable structures. This vortex shedding becomes the main cause of vibration, noise and periodic lift and drag on the cylinder. Norberg[1] compiled the theoretical, experimental and numerical results for a circular cylinder flow and analyzed the flow characteristic according to the Reynolds number. He showed that there was a considerable difference in characteristics between experimental and numerical results up to the present. As the Reynolds number increases, three dimensional instability increases in the wakes. Because of the pressure change, Karman vortex will occur periodically and the periodic characteristic of lift and drag will be generated [2]. Williamson [3] showed that two dimensional vortex shedding occurs when the Reynolds Number is less than 180. When it is greater than this value, he showed that unstable three dimensional vortex shedding will appear as Mode A and Mode B form[4].

In this paper the wake flows and aero- acoustic fields over a circular cylinders have been researched by high order, high resolution techniques that are used in two dimensional aero- acoustic analysis [5]. While theoretical equations and experiments were mainly used for the analysis of aero- acoustic in the past [6], a variety of research is being carried out for the mutual interference between flow and acoustics by using high order computation techniques in recent times. There are numerical instabilities, because a high order technique is a kind of central difference scheme.

The artificial dissipation model proposed by Kim&Lee [7] was applied to three dimensional flows to suppress the numerical instability. Numerical analysis using the high order technique requires a long computation time and a large memory capacity. Therefore, in order to improve the processing time, OpenMP parallel processing method was used. For the numerical result, the periodic characteristic of Strouhal Number due to vortex shedding was comparatively analyzed with other experiment values and two dimensional numerical results. The characteristics of second vortex and aero- acoustic fields were analyzed.

2. GOVERNING EQUATIONS AND NUMERICAL ANALYSIS METHOD

The non-dimensionalized three dimensional unsteady compressible Reynolds Averaged Navier-Stokes equation was transformed into a general coordinates, as Equation (1).

$$\frac{\partial \hat{Q}}{\partial t}+\frac{\partial \hat{E}}{\partial \xi}+\frac{\partial \hat{F}}{\partial \eta}+\frac{\partial \hat{G}}{\partial \zeta}=\frac{\partial \hat{E}_V}{\partial \xi}+\frac{\partial \hat{F}_V}{\partial \eta}+\frac{\partial \hat{G}_V}{\partial \zeta} \quad (1)$$

The cylinder diamerer (D), free stream velocity and free stream density are non-dimensional reference values, and, t,ξ,η and ζ are time and generalized coordinates. The symbols are referred to the reference literatures [8].

For a high order, high resolution numerical technique, OHOC (Optimized High-Order Compact) numerical technique used by Kim& Lee [7] in two dimensional aero-acoustic analyses was applied to three dimensional flows. For numerical differentiation, an implicit method using 7 node points was used, as shown in Equation (2). The coefficients are as shown in reference [5]. In order to maintain high resolution in time direction, a fourth order Runge-Kutta explicit method was used [9].

$$\beta f'_{i-2}+\alpha f'_{i-1}+f'_i+\alpha f'_{i+1}+\beta f'_{i+2}=\frac{1}{h}\sum_{m=1}^{3}a_m\left(f_{i+m}-f_{i-m}\right) \quad (2)$$

While it is possible to obtain high resolution with OHOC technique, dissipation and diffusion errors occur. These errors affect numerical stability significantly, because the wave characteristic cannot be accurately reproduced unlike an upwind scheme due to the characteristics of central difference method. In order to reduce this error and increase the stability of numerical scheme, the adaptive nonlinear artificial dissipation model proposed by Kim&Lee [7] was applied to the three dimensional flow.

The computation domain was set up to be 200 and $(\pi D/2, \pi D, 2\pi D)$ times the diameter D in the radial direction and the spanwise direction, respectively. The number of O-type grid system $201\times141\times(20,40,60)$ was used. Figure 1 shows the sample grid. In most of research for compressible flow, the wave reflection from the boundary is suppressed by using a non-reflecting characteristic method. However, for the high resolution technique, the wave reflection is suppressed with numerical dissipation by a wide area of buffer zone, because the wave reflection is not sufficiently suppressed even though a non-reflecting condition is used. In this paper, a wide numerical computation zone was set up and a coarse grid was used in the distant boundary region. . Therefore, while the numerical computation domain was set up wider than the methods with non-reflective characteristic condition, the boundary conditions were given very simply for non-reflecting condition.

For the parallel processing technique, the OpenMp method uses a shared memory based on a thread and allows relatively easy programming, the MPI method is carried out with independent memory in each processor with data communication, and the Cluster OpenMP has the advantages of both. OpenMP was mainly used in this research because of CPU time consumption for data communication

3. RESULT AND DISCUSSION

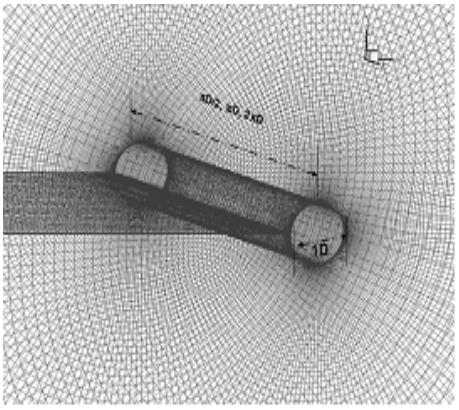


Fig. 1: *Three dimensional grid system*

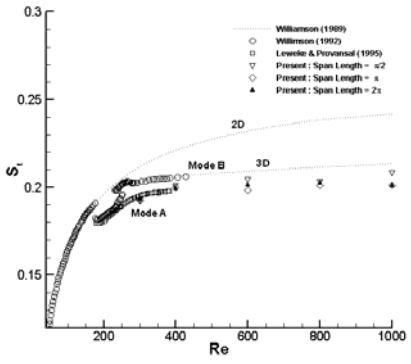


Fig. 2: *Strouhal number versus Reynolds number*

In Figure 2, the Strouhal number ($St = fD/U_\infty$) of lift coefficient was compared with other numerical results and experiment values, for the variations of Reynolds number and spanwise length. The Reynolds number ranges from

300 to 1000, which belongs to mode B referred by Williamson [3].

The results seriously differed from the 2-dimensional result of Williamson [3]. Even though the results in the range of mode B Reynolds number have a little difference with Williamson [3], they very closely agree with the Leweke [9][10][11]. There are no significant differences for the variation of spanwise length.

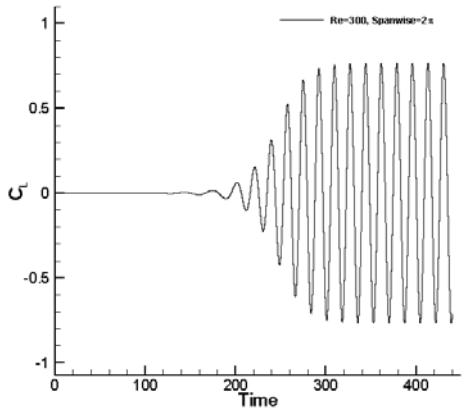


Fig. 3(a): *Time histories of lift coefficient at Re=300*

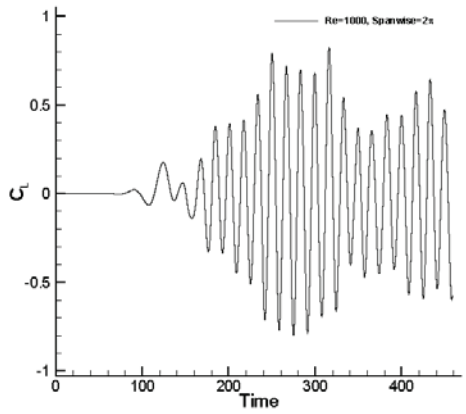


Fig. 3(b): *Time histories of lift coefficient at Re=1000*

Figures 3(a) and (b) shows the history of the lift coefficients for the Reynods number of 300 and 1000. Though the results of Reynolds number of 300 are converged to a periodic steady state, the results of Reynolds number of 1000 have other characteristics of low frequency for the three dimensional instability. In the other words, there are significant three dimensional effects in the case of Reynolds number of 1000, but there are no significant effects in the case of Reynolds number of 300.

Figure 4 shows the three dimensional Karman vortex disturbed by the secondary vortex. The strength of secondary vortex is very weak compared to the main Karman vortex. However, it can introduce flow instability as shown in Figure 3 of the history of lift coefficients.

Fig. 4: *3D karman vortex at Re=1000, spanwise length=2πD*

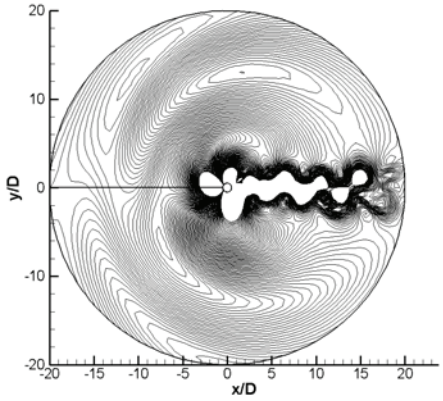


Fig. 5: *Visualization of dipole sound field on the center cross section of circular cylinder*

In Figure 5, pressure contours were drawn on the center cross section of z-axis in order to observe the propagation characteristic of acoustic waves for instance. From the figure, the numerical computation has been verified properly from the non-physical reflection, the propagation of pressure perturbation, and the periodic wake flow that is transmitted well even for long distances.

The pressure contours of Figures 6 show the propagation of sound waves and the Karman vortex shedding

simultaneously. Although the Karman vortices are slowly shedding with the pressure waves, the acoustic waves are propagated with a sound speed. The speed of propagation of acoustic waves, compared to the pressure waves, is very fast. The sound fields are affected by the three dimensional Karman vortices disturbed by the secondary vortices around the wake flow[12].

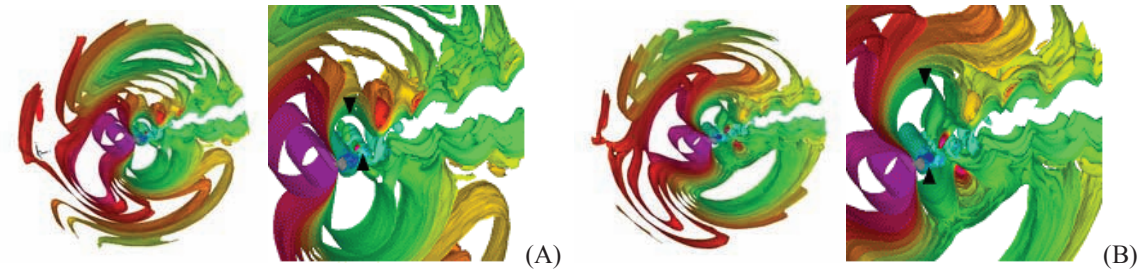


Fig. 6: *Visualization of dipole sound field and Karman`s vortex during one period*

4. CONCLUSIONS

The unsteady three dimensional compressible Reynolds averaged Navier Stokes equation was numerically computed to analyze the flow and aero acoustic fields past a circular cylinder by applying the OHOC technique that is a high order and high resolution numerical technique. To increase the computation speed for the large amount of numerical computations, a parallel processing method was used. Rather than the MPI method that requires massive data exchange between numerical domains, OpenMP that uses shared memory, was used. Since a non-reflective characteristic boundary condition needs a large buffer zone for a high order technique, it is not efficient. Therefore, in this research, while the numerical computation domain was set up wider than the methods with non-reflective characteristic condition, the very simple boundary conditions were used. Strouhal number of lift was compared with other two dimensional numerical computation results and experiment values. Three dimensional Karman vortex shedding characteristic disturbed by secondary vortices and the propagation characteristics of pressure waves and acoustic waves were analyzed

REFERENCES

1. Norberg, C. (2003). Fluctuating Lift on a Circular Cylinder: Review and new Measurements, *J. of Fluids and Structures*, Volume (17), 57-96
2. William K. Blake (1986). *Mechanics of Flow-Induced Sound and Vibration*, ACADEMIC PRESS. INC.
3. C.H.K. Williamson (1996). Three Dimensional wake transition, *J. Fluid Mech*., 328-345
4. T.LEWEKE and C.H.K. WILLIAMSON (1998). Three-dimensional instabilities in wake transition, *J. mech. B/Fluids*, volume (17) 571-586
5. J.W. Kim and D.J. Lee (1996) . Optimizd Compact Finite Difference Schemes with maximum Resolution, *AIAA Journal*, volume (34) number(5) 887-893
6. M. J. Lighthill (1952). On Sound Generated Aerodynamically: I.General Theory, *Proceedings of the Royal Society*, London, A221(1107) 564-587
7. J.W. Kim and D.J. Lee (1999). Adaptive Nonlinear Artificial Dissipation Model for Computational Aeroacoustics, *3rd CAA Workshop on Benchmark Problems*, USA, November
8. C.H. Woo, J.S. Kim, and K.H. Lee (2006) . Analysis of Two- and three-Dimensional Supersonic Turbulence Flow around Tandem cavities, *Journal of Mechanicals Science and Technology(KSME Int.J.)*, volume (20) number (8) 1256~1265
9. K.C. Hoffmann, and S.T. Chiang (1993). Computational Fluid Dynamics for Engineers, *Engineering Education System*, USA
10. A. Roshko (1955), On the wake and drag if bluff bodies, *J.Aeronaut.Science*., volume(22) 124
11. U. Fey, M. Koenig and H. Eckelmann (1998). A new Strouhal-Reynolds number relationship for the circular cylinder in the range $47 < \mathrm{Re} < 2\times10^5$, *Phys. Fluids*, volume(10) 1547
12. R. Franke, W. Rodi, and B. Schonung (1990). Numerical Calculation of Laminar Vortex Shedding Flow Past Cylinders*, J. Wind Engineering and Industrial Aerodynamics*, volume(35) 237-257

Aerodynamic Database Generation Using Surrogate Model-Based Adaptive Sampling and Automated Mesh Refinement

Andrea Nelson,* Matthew McMullen,†

Abstract submitted to the 21st International Converence on Parallel Compuatational Fluid Dynamics,
May 18–22, 2009, Moffett Field, CA

I. Introduction

THE design of modern flight vehicles requires the characterization of aerodynamic parameters over a mission flight envelope. Currently this process involves choosing key design sites representative of the mission and performing individual analyses that are combined to form an aerodynamic database. The database is then inspected for inaccurate solutions and regions where more information might be necessary and iterated upon. This process is costly both in man hours and computational hours and lacks parallel efficiency due to its iterative nature. In this work we propose a framework that both automatically creates data in any area of the mission envelope that conforms to a constant error threshold, and intelligently selects points to populate the characteristic database. This framework includes two encapsulated modules. The first is a flow solver with a solution adaptive mesh process, which provides uniform error levels across data in an automated fashion. The second module is a Kriging surface generator, which fits a Kriging model to a given set of data and performs adaptive sampling iterations based on local modeling error and predicted improvements in the subsequent surface. There are multiple levels of parallelism within the two modules, creating an efficient and scalable database generation process.

The roadblock to automating computational fluid dynamics (CFD) analyses over large mission envelopes is typically the volumetric grid generation. Historically, users of CFD solvers have attempted to manually develop a single grid for database generation. Using refinement studies based on a subset of the operating conditions, the grid would attempt to capture all possible features that can be exhibited within the design space. Classical theory surrounding partial differential equations predicts their discretization error as a function of the grid spacing and higher order derivatives in the solution. Combining these predictions with practical experience, CFD users have refined grids around features exhibiting singificant gradients including trailing/leading edges and shock fronts. The existence, location and impact of these flow features on the figure of merit will vary significantly through the design space. By design, this gridding approach over-resolves large regions of the flow field for any specific case, but attempts to address the resolution requirements for the database as a whole. There are many drawbacks to this approach. First, this approach pays a significant computational cost in attempting to employ a grid capable of resolving all potential flow features. Second, the cost of this over refinement could make complex problems intractable and may result in a database with varying discretization error levels throughout its space.

An alternative to this approach involves users manually performing grid resolution studies at each operating location within the database. This approach has the drawback of incurring costs in terms of user intervention and can be cost prohibitive on large databases.

This paper employs a mesh adaptation approach that addresses all of the aforementioned issues. The adaptation process uses an adjoint formulation which provides error estimates for a user defined functional. Within the process, a sequence of grids are automatically refined to minimize this error. The process can

*Research Scientist, ELORET Corporation, Andrea.J.Nelson@nasa.gov
†Research Scientist, ELORET Corporation, Matthew.S.McMullen@nasa.gov

be stopped when a requisite error level has been reached, resulting in a database based on functionals with uniform error levels over the design space.

Chosing locations for CFD solutions within a flight vehicle's mission envelope with no knowledge of the design space apriori can result in the under resolution of key features and the over resolution of regions with no important features. However, iterating on a database by hand is time consuming and inefficient. Leveraging the utility of optimization-based surrogate model tools introduces the ability to intelligently and automatically select analysis locations based on the overall mission. These database filling tools can give users more information from limited computational resources, and can utilize large sets of processors by running multiple analyses in parallel.

The surrogate modeling tool presented in this paper utilizes a Kriging model with adaptive sampling. The framework incorporates error measures output from the Kriging predictor to adaptively sample data in locations with a statistical likelihood to improve the subsequent Kriging surface. Coupled with the adaptive mesh flow solver, the database generation process is automated and entirely scalable to utilize large sets of computational resources.

II. Framework Implementation

II.A. Solution Adaptive Meshes

The figures of merit used in this paper will be based on functionals of surface pressure. CART3D, a cartesian mesh based inviscid flow solver will provide the flow and adjoint solutions as well as a process for the grid adaptation.[1,2] Based on adjoint and residual information, the grid adaptation process will minimize the error in a functional that is also the database figure of merit. The adaptation process stops when an error tolerance has been met, resulting in a figure of merit with a relatively uniform discritization error over the extents of the domain.

Nemec et. al. have described an adaption strategy that employs a decreasing threshold with the cycle count to reduce computational cost in comparison to fixed threshold strategies.[2] The challenge to this research is to define an adaptation threshold not from a sense of efficiency but rather from a robustness perspective. Specifically, the flow and adjoint solutions are a function of both residual and discretization error. An adaptation producing a grid minimizing discretization error at the expense of no longer supporting a steady flow is not useful. At this point in the development of these tools, users will need to understand the limit in grid refinement (throughout the flow field) that no longer supports steady flow and set the error thresholds appropriately.

II.B. Kriging Framework

A Kriging surface, first developed in the geostatistics community,[3–5] models data based on the assumption that the responses of a set of data points are the outcome of a stochastic process. Given a response vector Y, the Kriging model equations follow

$$Y = F\beta + e \tag{1}$$

where F is a matrix of underlying regression functions and e is the white noise of a stochastic process. The determination of e is based on the following equation

$$E[e] = 0, \quad E[ee^T] = \sigma^2 R$$

where $E[.]$ deontes the expected value, and R is the covariance matrix given by a chosen correlation function. The Kriging model is created by solving for β and e with a generalized least squares fit given F and R.

The figure of merit used in this work to track the convergence of the Kriging model will be the compact expression of cross-validated error derived by Mitchell and Morris.[6] Although referred to as the Cross-Validated Root Mean Squared Error (CVRMSE) by Mitchell and Morris, we will simply call it the predicted Cross-Validated Error (CVE).

II.C. Kriging Based Adaptive Sampling

The focus of adaptive samping is to generate a set of candidate points to be analyzed in an intelligent manner. There are a number of ways to do this. In this work two different methods of sampling points will

be demonstrated. The first method utilizes the Expected Improvement Function, while the second uses the predicted error output from the Kriging model.

The first method of adaptive sampling was developed for optimzation problems where the two main objectives when selecting new points are improvement of the model and prediction of new extrema. The objectives can be described mathematically with the Expected Improvement Function (EIF).[7] Given a kriging prediction $\hat{y}$, and standard error s, the EIF is given by the following:

$$EIF(x) = (y_{min} - \hat{y})\Phi(\frac{y_{min} - \hat{y}}{s}) + s\phi(\frac{y_{min} - \hat{y}}{s}) \qquad s > 0 \tag{2}$$

$$= 0 \qquad s = 0 \tag{3}$$

where y_{min} is the minimum value of the current dataset, Φ is the standard normal distribution function and ϕ is the standard normal density function. The first term above causes an increase in the EIF in areas where there is a predicted extrema, while the second term causes an increase in the EIF in areas where the model uncertainty is large.

The second method of adaptive sampling simply seeks to minimize the global predicted error of the Kriging surface by seeking the location of its maximum and adding more data in that location. The Kriging predicted error s is given by the following equation

$$s^2(x^*) = \sigma^2[1 - r'R^{-1}r + \frac{(1 - \mathbf{1}'R^{-1}r)^2}{\mathbf{1}'R^{-1}\mathbf{1}}]. \tag{4}$$

where $\mathbf{1}$ is a vector of ones, R is the covariance matrix of the Kriging model and r is the correlation vector of the point at which the error is being predicted.

Finding the location where either the EIF or predicted error are maximized would require a separate optimization loop for each adaptive sampling iteration. We instead select a number of candidate points chosen via a hypercube binning algorithm based on the current distribution of points. The figure of merit (EIF or predicted error) of each of these candidate points is calculated and the points with the highest figure of merit are chosen for evaluation.

II.D. Parallel Framework

There are multiple levels of parallelism within this work. At the solver level, CART3D (as well as most modern CFD solvers) is based on a flow solver employing a parallel communciation paradigm, in this case using OpenMP. At the functional evaluation level, the database tools interface with the computers batch queuing system to allow multiple flow solvers to simultaneously execute. At the database modeling level, this set of tools implements an adaptive sampling algorithm that is parallelized using MPI to distribute the computation of the WEIF for a batch of candidate sampling locations.

III. Problem Formulation

The framework being presented in this paper is in its preliminary stages and as such we have chosen a standard model problem as a proof of concept. More complex geometries will be presented in the paper. The model problem presented here is a transonic wing based on the NACA0012 airfoil. The functional being modeled is the ratio of lift to drag at subsonic and supersonic Mach numbers, and a range of angles of attack. The error tolerance for the adaptive meshing algorithm is 0.01. The framework was initialized with a dataset of 30 points chosen with a Latin Hypercube algorithm. The final databases were generated using batches of 10 adaptively sampled points per iteration over 7 iterations for each adaptive sampling algorithm.

IV. Results and Discussion

The Kriging surfaces constructed using the final databases are shown in Figs. 1 and 2 for the WEIF and predicted error adaptive sampling methods, respectively. Each Kriging surface uses a zero-order polynomial as a regression function and an exponential correlation function to minimize the predicted CVE. The final predicted error of the resulting Kriging surfaces are shown in the contour plots in Figs. 6 and 4, along with the locations of the adaptively sampled points for each method. The distribution of points in Figs. 6 and 4 illustrate the differences in the two methods. With well-behaved functions, the largest Kriging predictive

errors will generally occur near the boundaries of the domain, and for hypercube shaped domains, will be maximum in the corners. Thus, the predictive error driven sampling method will tend to cluster points in the corners when the surface is well characterized by a small number of points. The WEIF based sampling method, however, takes into account looking for new extrema, leading to a clustering of points in the corners were current extrema are located. These trends may not be apparent in modeling more complicated functions.

The maximum predicted errors for the WEIF adaptive sampling and the predicted error adaptive sampling are 0.0015 and 0.0010, respectively, which are lower than the error tolerance of the data produced from the flow solver. The convergences of the predicted CVE and predicted error from the two methods are shown in Figs. 5 and 6, respectively. The two measures of error follow the same convergence trend, with neither method of sampling exhibiting superior behavior. The surface being modeled is simple enough that either sampling method is sufficient to fully characterize the output.

V. Summary

In the full paper we will demonstrate the creation of an aerodynamics database for several problems. Our first model problem was a NACA 0012 wing. Subsequent problems will be a supersonic transport and the Orion launch abort vehicle at mach numbers and angles of attack that encompass flight conditions.

The extensive variation in geometry and operating conditions will demonstrate the validity and robustness of this approach for problems of industrial interest. Furthermore, we will demonstrate the efficiency gains of this approach over classical techniques.

References

[1]Melton, J. E., Berger, M. J., Aftosmis, M. J., and Wong, M. D., "3D Applications of a Cartesian Grid Euler Method," AIAA Paper 1995–0853, Aerospace Sciences Meeting and Exhibit, Reno, NV, Jan. 1995.

[2]Nemec, M., Aftosmis, M. J., and Wintzer, M., "Adjoint-Based Adaptive Mesh Refinement for Complex Geometries," AIAA Paper 2008-0725, Aerospace Sciences Meeting and Exhibit, Reno, NV, Jan. 2008.

[3]Matheron, G., "Principles of geostatistics," *Economic Geology*, Vol. 58, 1963, pp. 1246–1266.

[4]Cressie, N., "The Origins of Kriging," *Mathematical Geology*, Vol. 22, 1990, pp. 239–252.

[5]Cressie, N., *Statistics for Spatial Data*, John Wiley, New York, 1993.

[6]Mitchell, T. J. and Morris, M. D., "Bayesian Design and Analysis of Computer Experiments: Two Examples," *Statistical Sinica*, Vol. 2, 1992, pp. 359–379.

[7]Jones, D. R., Schonlau, M., and Welch, W. J., "Efficient Global Optimization of Expensive Black-Box Functions," *Journal of Global Optimization*, Vol. 13, 1998, pp. 455–492.

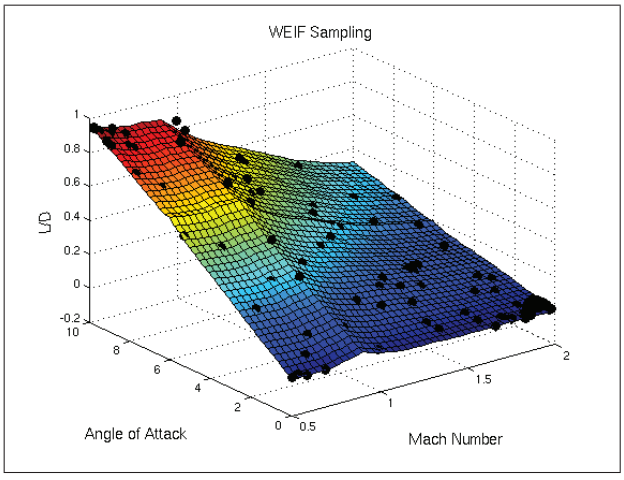


Figure 1. Final Kriging Surface, WEIF-based Sampling

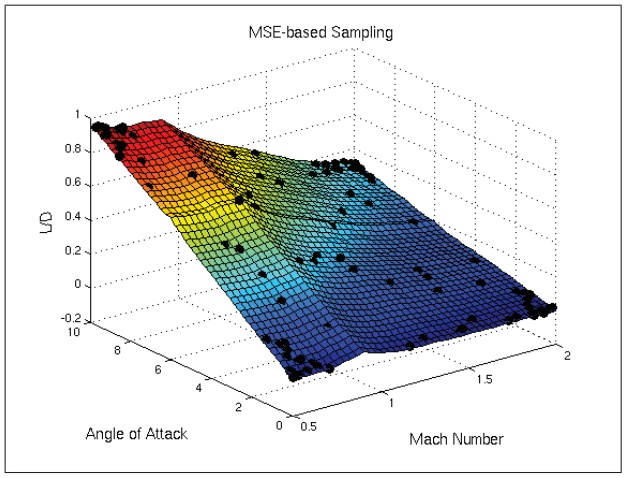


Figure 2. Final Kriging Surface, predicted error-based Sampling

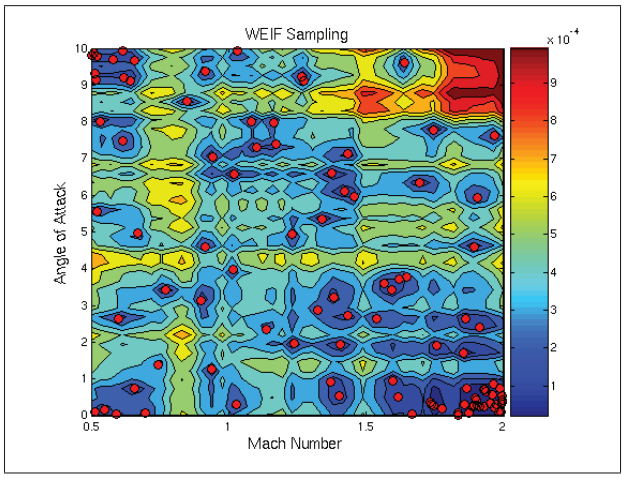


Figure 3. Predicted Error Contour, WEIF-based Sampling

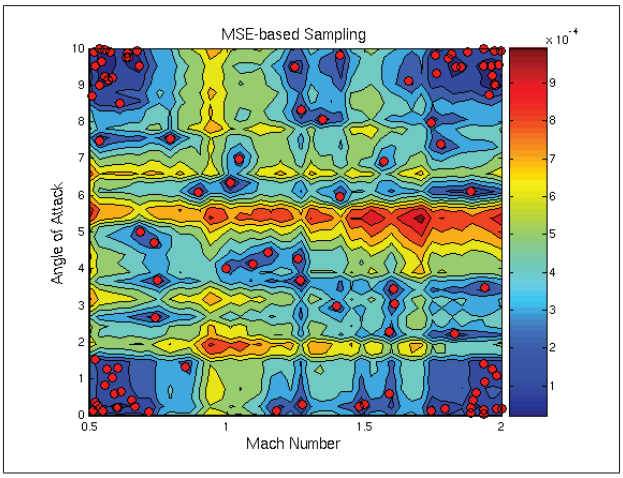


Figure 4. Predicted Error Contour, predicted error-based Sampling

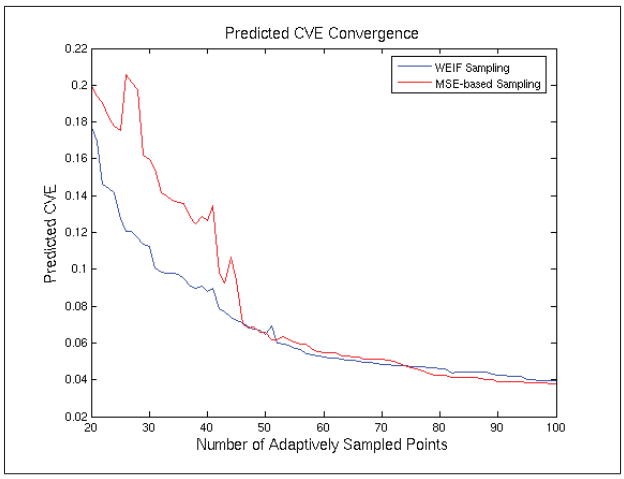


Figure 5. Predicted Cross-Validated Error vs. Adaptive Sampling Points

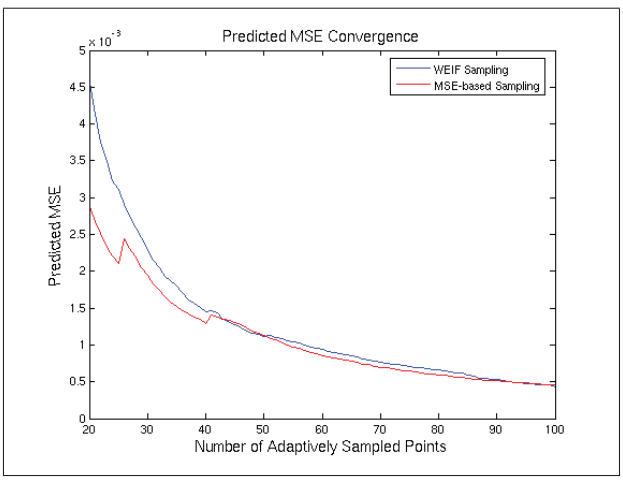


Figure 6. Kriging Predicted Error vs. Adaptive Sampling Points

Parallel Time-Accurate Computations of Dynamic Derivatives

Jubaraj Sahu
U.S. Army Research Laboratory
AMSRD-ARL-WM-BC
Aberdeen Proving Ground, Maryland 21005-5066, USA
E-mail: sahu@arl.army.mil

KEY WORDS: Time-Accurate, Unsteady Flows, Dynamic Derivatives, Multidisciplinary Modeling

ABSTRACT

This abstract describes a multidisciplinary computational study undertaken to compute the free-flight aerodynamics including the dynamics derivatives of projectiles. Advanced computational capabilities both in computational fluid dynamics (CFD) and rigid body dynamics (RBD) have been successfully fully coupled on high performance computing (HPC) platforms for "Virtual Fly-Outs" of munitions similar to actual free flight tests in the aerodynamic experimental facilities. Time-accurate Navier-Stokes computations have been performed to compute the unsteady aerodynamics associated with the free flight of projectiles using an advanced scalable unstructured flow solver on a highly parallel Linux Cluster. Some results relating to the portability and the performance of the flow solver on the Linux clusters are also addressed. Time-accurate numerical techniques include both the "virtual fly-out" and "virtual wind tunnel" techniques. All aerodynamic force and moment coefficients including the dynamic damping derivatives have been extracted from the fully coupled CFD/RBD numerical solutions. Time-accurate CFD has been used separately to compute the dynamic pitch-damping derivatives using the virtual wind tunnel technique. Computed dynamic pitch-damping derivatives using this virtual wind tunnel method have been compared with those obtained from the fully coupled virtual fly-out approach and flight tests.

COMPUTATIONAL METHODOLOGY

As part of a Department of Defense High Performance Computing (HPC) grand challenge project, the U.S. Army Research Laboratory (ARL) has recently focused on the development and application of state-of-the art numerical algorithms for large-scale simulations [1,2] to determine both steady and unsteady aerodynamics of projectiles with and without flow control. Fully time-accurate CFD/RBD coupled methods do offer the greatest potential to provide the potential for accurate and simultaneous prediction of all aerodynamic coefficients including the dynamic derivatives (roll-damping, pitch-damping, and Magnus moments). Our interest in the fully coupled techniques [3,4] is to use them to compute the flight trajectory of a projectile and fly it through the supercomputers similar to what happens with the actual free-flight of the projectile inside an aerodynamics experimental facility.

The complete set of three-dimensional (3-D) time-dependent Navier-Stokes equations [5] is solved in a time-accurate manner for simulations of unsteady flow fields associated with both spinning and finned projectiles during flight. The 3-D time-dependent Reynolds-averaged Navier-Stokes (RANS) equations are solved using the finite volume method [6,7]: Second-order discretization was used for the flow variables and the turbulent viscosity equations. Two-equation [8] and higher order hybrid RANS/LES [9] turbulence models were used for the computation of turbulent flows. Grid was actually moved to take into account the spinning motion of the projectile and grid velocity is assigned to each mesh point. This general capability can be tailored for many specific situations. For example, the grid point velocities can be specified to correspond to a spinning projectile. In this case, the grid speeds are assigned as if the grid

is attached to the projectile and spinning with it. Similarly, to account for rigid body dynamics, the grid point velocities can be set as if the grid is attached to the rigid body with six degrees of freedom (6 DOF). For the rigid body dynamics, the coupling refers to the interaction between the aerodynamic forces/moments and the dynamic response of the projectile/body to these forces and moments. The forces and moments are computed every CFD time step and transferred to a 6DOF module which computes the body's response to the forces and moments. The response is converted into translational and rotational accelerations that are integrated to obtain translational and rotational velocities and integrated once more to obtain linear position and angular orientation. The grid point locations and grid point velocities are set from the dynamic response.

For determination of dynamic derivatives, time-accurate CFD can be used separately to compute the dynamic derivatives using the virtual wind tunnel technique which really is a special case of the coupled method with translational motion set to zero. In this case, all calculations are done only in the uncoupled mode with the zero translational velocity. The spin component of the projectile or the angular velocity of the projectile is added to compute the rolling motion of the projectile. With the addition of spin, time-accurate calculations are performed for a few cycles of spin until converged periodic forces and moments are obtained. A sufficient number of time steps are similarly performed for the angular pitching motion case where the pitching motion (sinusoidal, for example) is imposed.

Our interest here is in the numerical prediction of dynamic derivatives of projectiles using time-accurate viscous Navier-Stokes techniques and unstructured grids and in particular, for the pitch damping moment coefficient. The pitch damping moment coefficient contains two parts; one part proportional to transverse angular velocity (pitching velocity, q), and a second part proportional to the rate of change of total angle of attack ($\dot{\alpha}$).

$$Pitch\ Damping\ Moment = \frac{1}{2}\ \rho\ V^2\ S\ \ d\left(\frac{q\,d}{2V}\right)(C_{m_q} + C_{m_{\dot{\alpha}}}) \qquad (2)$$

where C_{m_q} is the pitch damping moment coefficient due to q, $C_{m_{\dot{\alpha}}}$ is the pitch damping moment coefficient due to $\dot{\alpha}$, ρ is the air density, V is the projectile velocity, S is the projectile reference area, and d is the projectile reference diameter. The pitch damping coefficient sum, $(C_{m_q} + C_{m_{\dot{\alpha}}})$ is obtained from an oscillating forced motion of the projectile with respect to the center of gravity. The imposed motion is sinusoidal and is given by a simple function,

$$\alpha = \alpha_m + \alpha_0 \sin(\omega t) \qquad (3)$$

where, α is the instantaneous angle of attack, α_m is the mean angle of attack, α_0 is the pitch amplitude, and ω is the frequency of oscillation. With proper normalization, it can be written in terms of a reduced frequency, k which is given by k = q d/ 2V.

The roll damping moment is calculated from the unsteady rolling motion of the projectile. In this case, projectile spins and time-accurate calculations are carried out with the body moving until the rolling moment is converged for a given constant spin rate.

PARALLEL COMPUTATIONAL ASPECTS

The CFD++ computational fluid dynamics simulations software runs on a wide variety of hardware platforms and communications libraries including MPI, PVM, and proprietary libraries of nCUBE, Intel Paragon etc. Inter-CPU communications are included at the fine grid level as well as all the multigrid

levels to help ensure high degree of robustness consistently observed in using CFD++, independent of the number of CPUs being employed.

The flow solver can be run on any number of CPUs in parallel. The mesh files and the restart files that are needed/generated for single CPU runs are identical to those associated with multi CPU runs. For multi-cpu runs, a domain-decomposition file is needed which defines the association between cell number and CPU. The software suite includes several domain decomposition tools and it is also fully compatible with the METIS tool developed at the University of Minnesota. The code runs in parallel on many parallel computers including those from Silicon Graphics, IBM, Compaq (DEC and HP), as well as on PC workstation clusters. Excellent performance (see Figure 1 for the timings on a 12-million mesh) has been observed up to 128 processors on IBM SP P3 (375 MHz) and especially on IBM SP P4(1.7GHz), and Linux PC cluster (3.06 GHz).

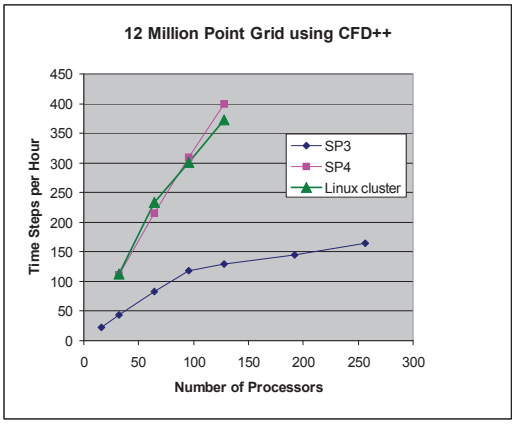
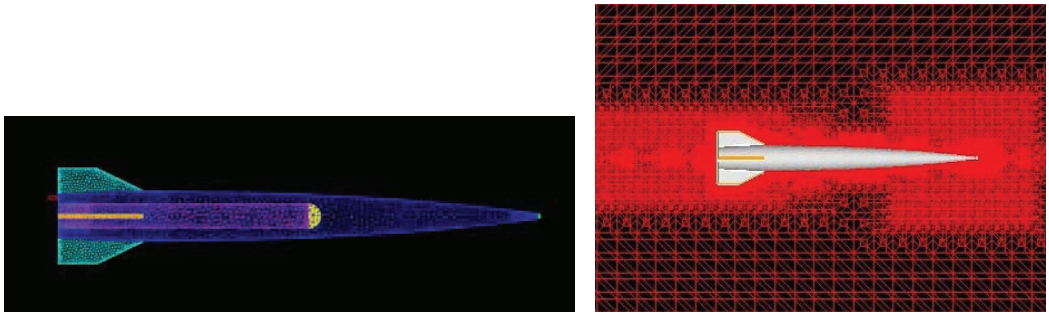


Figure 1. Parallel Speedups (12-million grid).

RESULTS

Time-accurate numerical computations were performed using both virtual fly-out and virtual wind tunnel approaches for a fin-stabilized projectile. The dynamic derivatives, the pitch damping moment and the roll damping moment coefficients were obtained from these solutions. Specifically, uncoupled, unsteady CFD procedures were used to calculate the pitch-damping moment coefficients using sinusoidal pitching motion of the projectile.



(a) (b)

Figure 2. (a) Surface mesh and (b) unstructured mesh near the finned projectile,.

The supersonic projectile modeled in this study is an ogive-cylinder-finned configuration. The length of the projectile is 121 mm and the diameter is 13mm. The ogive nose is 98.6 mm long and the afterbody has a 22.3 mm, 2.5° boat-tail. Four fins are located on the back end of the projectile. Each fin is 22.3 mm long and 1.02 mm thick. An unstructured computational surface mesh was first generated for this projectile (see Figure 2a). The full grid generated was also unstructured (see Figure 2b) and the total number of grid points was about 6.5 million for the full grid. The first spacing away from the wall was selected

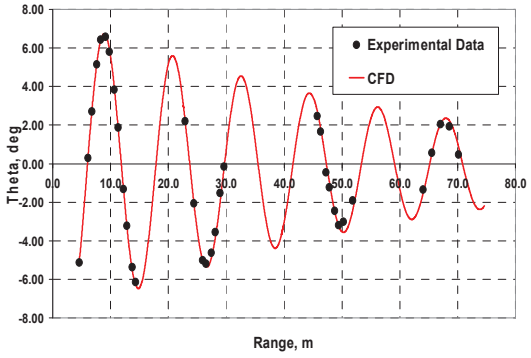


Figure 3. Euler pitch angle vs. x-distance.

to yield a y+ value of 1.0. In general, most of the grid points were clustered in the boundary-layer, near the afterbody fin, and the wake regions. The projectile configuration has a base cavity (see Figure 2a) and was included in the mesh generation process using MIME [10].

Using the coupled CFD/RBD virtual fly-out method, numerical computations have been made for the finned body at an initial velocity of 1034 m/s. The initial angle of attack was, $\alpha = 4.9^{o}$ and initial spin rate was 2500 rad/s. The orientation and the position of the projectile of course change from one instant in time to another as the projectile flies down range. Figure 3 shows the variation of the Euler pitch angle with distance traveled. As seen in this figure, both the amplitude and frequency in the Euler angle variation are predicted very well by the computed results and match extremely well with the data from the flight tests. One can also clearly see that the amplitude damps out as the projectile flies down range i.e. with the increasing x-distance. The results produced by the virtual fly-out simulations provide the total aerodynamic forces and moments at every time step as the projectile flies down range. For a variety of reasons, one may want to extract the traditional aerodynamic force and moment coefficients from these coupled CFD/RBD simulations. One way is to feed the CFD/RBD generated data back into software such as ARFDAS [11] and back out the aerodynamic coefficients with the same procedure used on the actual test data. The extracted aerodynamic coefficients are compared with the same coefficients obtained using the actual test data. The ARFDAS fitting procedure produces aerodynamic forces that match well with the computed results from the virtual fly-out of the projectile.

The unsteady procedure in the virtual wind tunnel approach use the pitching and the rolling motions and provide not only the dynamic derivatives, but also all the traditional static aerodynamic force and moment coefficients. For the finned projectile, emphasis is put on the numerical computation of the pitch damping moment coefficient and the roll damping moment derivative. Sinusoidal pitching motion is imposed and time-accurate CFD computations have been performed at various supersonic velocities. Figure 4 shows the time-history of the pitching moment coefficient as a function of angle of attack at M=1.6. Computed pitching moment coefficients shown here correspond to first two complete pitch cycles. Each full cycle of imposed pitching motion includes both the pitch-up and pitch-down parts. Other than the initial startup differences, the computed results converge quite rapidly at this velocity. Results obtained for the third cycle (not shown here) was virtually identical to that of the second cycle. The pitch damping moment coefficient is related to the difference in the values of the pitching moment coefficients at zero degree angle of attack in this case between the up and down portions of the imposed pitching motion.

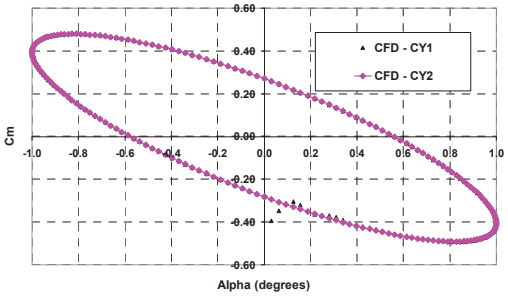


Figure 4. Time history of the pitching moment coefficient with angle of attack in the pitch cycle, M = 1.6.

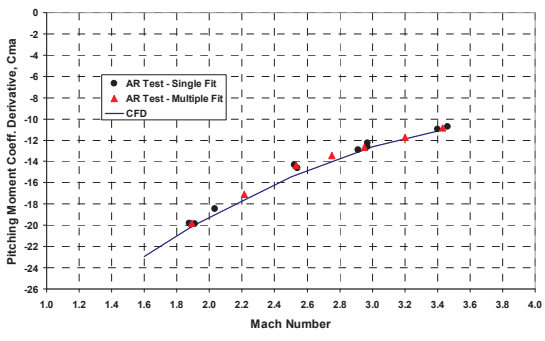


Figure 5. Pitching moment coefficient vs. Mach number.

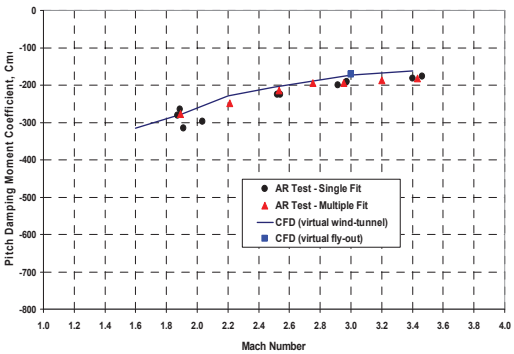


Figure 6. Pitch damping moment coefficient vs. Mach number.

Computed static pitching moment coefficients as well as computed dynamic pitch damping moment coefficients were obtained from a series of time-accurate calculations at different supersonic velocities from M=1.6 to M=3.4 and are shown in Figures 5 and 6, respectively. These computed results are compared with the data derived from free flight tests for the same projectile configuration using single and multiple fits. Figure 5 shows the variation of the computed pitching moment coefficient with Mach number. As shown in this figure, the static pitching moment coefficient increases with Mach number from M = 1.6 to M = 3.4 and the computed results match very well with the test results. Figure 6 shows the variation of the dynamic pitch damping moment coefficient with Mach number. Again, computed pitch damping moment coefficients have been compared to those obtained from the flight tests for the same configuration and the same supersonic velocities. The computed results are generally in good agreement with the data and they are within the accuracy of the experimental test results. The full paper will include results obtained for the roll damping derivatives. Also, additional computations are in progress for a spinning projectile at a transonic speed and the full paper will include computed results for this case as well.

CONCLUDING REMARKS

This abstract describes a computational study undertaken to compute the free-flight aerodynamics and in particular, the dynamic pitch-damping derivatives of finned projectiles at supersonic speeds. Fully three-dimensional time-accurate numerical techniques include both the "virtual fly-out" and "virtual wind tunnel" techniques. Virtual fly-out simulations were performed using the coupled CFD and RBD methods. Computed Euler pitch angles match very well with the data obtained from the range tests for the same projectile at a supersonic speed. All aerodynamic force and moment coefficients including the dynamic damping derivatives have been extracted from the fully coupled CFD/RBD numerical solutions. Time-accurate CFD has been used separately to compute the dynamic derivatives using the virtual wind tunnel technique. Computed dynamic pitch-damping derivatives obtained using this virtual wind tunnel method have been compared with those obtained from the fully coupled virtual fly-out approach and flight tests and are all in good agreement. This work demonstrates a coupled method to accurately predict the time-accurate unsteady aerodynamics of projectiles and provides for a new way to obtain all aerodynamic coefficients including the dynamic pitch-damping and roll damping derivatives.

REFERENCES

1. J. Sahu, Pressel, D., Heavey, K.R., and Nietubicz, C.J., "Parallel Application of a Navier-Stokes Solver for Projectile Aerodynamics", Proceedings of Parallel CFD'97 Meeting, Manchester, England, May 1997.
2. J. Sahu, Edge, H.L., Dinavahi, S., and Soni, B., "Progress on Unsteady Aerodynamics of Maneuvering Munitions" Users Group Meeting Proceedings, Albuquerque, NM, June 2000.
3. Sahu, J., "Time-Accurate Numerical Prediction of Free Flight Aerodynamics of a Finned Projectile" ARL-TR-3603, Aberdeen Proving Ground, MD, September 2005.
4. Sahu, J., "Time-Accurate Computations of Free-Flight Aerodynamics of a Spinning Projectile With and Without Flow Control" ARL-TR-3919, APG, MD, September 2006. 32.
5. T. H. Pulliam and J. L. Steger, "On Implicit Finite-Difference Simulations of Three- Dimensional Flow" *AIAA Journal*, vol. 18, no. 2, pp. 159–167, February 1982
6. Peroomian, O. and Chakravarthy S., "A `Grid-Transparent' Methodology for CFD," AIAA Paper 97-0724, Jan. 1997.
7. O. Peroomian, S. Chakravarthy, S. Palaniswamy, and U. Goldberg, "Convergence Acceleration for Unified-Grid Formulation Using Preconditioned Implicit Relaxation." AIAA Paper 98-0116, 1998.
8. U. Goldberg, O. Peroomian, and S. Chakravarthy, "A Wall-Distance-Free k-ε Model With Enhanced Near-Wall Treatment" *ASME Journal of Fluids Engineering*, Vol. 120, 457-462, 1998
9. P. Batten, U. Goldberg and S. Chakravarthy, "Sub-grid Turbulence Modeling for Unsteady Flow with Acoustic Resonance", AIAA Paper 00-0473, *38th AIAA Aerospace Sciences Meeting*, Reno, NV, January 2000
10. Chakravarthy, S., "MIME ", Metacomp Technologies, Inc., Agoura Hills, CA, 2006.
11. Arrow Tech Associates. "ARFDAS Technical Manual." South Burlington, VT, 2001.

Exploring Discretization Error in Simulation-Based Aerodynamic Databases

Michael J. Aftosmis[a] and Marian Nemec[b]

[a]**NASA Ames Research Center**,
Mail Stop T-27B, Moffett Field, CA 94035, USA

[b]**ELORET Corp.**,
Mail Stop T-27B, Moffett Field, CA 94035, USA

keywords: parallel, parametric studies, error, database, adjoint, Cartesian, Cart3D, CFD

Abstract

The proposed paper examines the level of discretization error in simulation-based aerodynamic databases and introduces strategies for error control in these databases. Simulations are performed using a parallel, multi-level Euler solver on embedded-boundary Cartesian meshes. Discretization errors in user-selected outputs on a given mesh are estimated using the method of adjoint-weighted residuals and we use adaptive mesh refinement to reduce these errors below user-specified tolerances. Using this framework, we compare the cost and accuracy of three approaches for aerodynamic database generation. In the first approach, all cases in the database are computed on a fixed mesh – the *de facto* standard of aero-database generation. The second approach uses a uniform error tolerance, while a third approach uses a relative error tolerance linked to the output value. We quantitatively assess the error landscape for all three methods. This investigation provides insight into the sensitivity of the database to a variety of sources, such as the presence of shocks in the flow and the stiffness of the governing equations at sonic conditions or near the incompressible limit. The results show that such pathologies may cause variations in mesh sizes spanning one to two orders of magnitude, and highlight a significant weakness of the fixed-mesh approach. We propose hybrid strategies that minimize simulation cost in sensitive regions of the database and quantitatively explore their use.

1. Introduction

THE SHIFT to multi-core CPU architectures has rapidly accelerated the growth of supercomputing resources.[1] This marked increase in the level of high-performance computing now offers both unprecedented capability and capacity to the aerodynamic simulation community.[2] These systems are capable of aerodynamic simulations with 10^8-10^{10} degrees of freedom, offering ever increasing physical fidelity.[3,4] While such extremely large "capability" simulations are becoming commonplace, the engineering community has focused on the enormous capacity of these systems through an increasing reliance on parametric, trade and optimization studies. In industrial settings, the main role of high-end computing is performance database generation. Aerodynamic databases with 10^3-10^4 simulations have become common and "production CFD" is the mainstay of the workload on this hardware.

The quality of these databases hinges on the level of the discretization error associated with the simulation outputs. Simply put, how much does the mesh influence the results? In this work, we use the method of adjoint-weighed residuals to estimate the discretization error in each simulation and examine the resulting error distribution. Adaptive mesh refinement is used for error control in each case of the database. These investigations use the parallel multi-level Cartesian Euler solver developed in references [5] and [6] to produce the aerodynamic data. This simulation package has been used extensively on large shared and distributed memory systems and has very good parallel scalability.[2,3,7] The robustness and automation of this simulation package has led to its wide adoption for producing aerodynamic databases in support of engineering analysis and design.[8,9,10,11]

The use of adjoint-based error estimates and adaptive mesh refinement in database construction increases the computational cost of each data point because several flow and adjoint solutions are required to generate a mesh that satisfies the user-specified output tolerances. The adjoint-solver, sensitivity calculation, and error-estimator all use the same parallel, multilevel framework as the base Euler solver and all achieve the same high level of parallel efficiency.[12,13,14]

In addition to adaptively meshing a simulation, the error-estimation module can also be used in a single-pass mode to assess the error in a specific functional on a given input mesh. Provided that the Euler simulation on this mesh is sufficiently good, this method allows us to accurately assess the discretization-error in a specific output on a particular mesh. This approach can be applied to each simulation in an aerodynamic database, yielding an *a-posteriori* estimate of the error landscape for that database.

Our investigations employ both of these techniques to investigate the role of discretization error in simulation-based aero data. We begin by briefly reviewing the salient features of adjoint-error estimation using a discrete adjoint solver. We then present both fixed-mesh and error-controlled databases spanning incompressible, transonic and supersonic flow. Our analysis tracks both error and cost in these databases and aims to quantitatively understand the implications of both strategies.

2. Error Assessment and Estimation

Figure 1 shows two sketches showing convergence of a typical aerodynamic output (C_L, C_D, etc.) with mesh refinement. In the sketch at the left of figure 1, E is the total error in this output due to discretization-error in the numerical solution. This error is defined as the difference between the exact functional value, J, and the value obtained when evaluating this functional using the discrete solution on the current mesh, $J(U_H)$. Rather than attempt to compute E directly, we follow the approach of Ref. [16], and consider instead the simpler problem of estimating how our discrete evaluation $J(U_H)$ would change if we solved on a finer mesh, h. The *relative error*, e, is sketched at the right of figure 1, and is defined as the difference between the functional evaluations on the current mesh, H, and the finer, embedded mesh h.

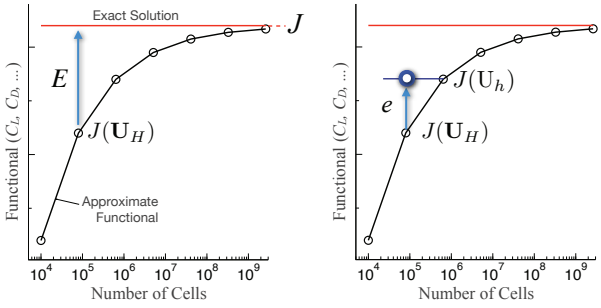


Figure 1: Convergence of functional J with mesh refinement showing the definition of total error, E (left) and relative error, e (right).

$$e = |J(U_H) - J(U_h)| \qquad (1)$$

For a second-order method on a sufficiently smooth solution in the asymptotic range, knowing the relative error gives the total error in the output.

$$E = e + \frac{1}{4}e + \frac{1}{4^2}e + \cdots \quad = \frac{4}{3}e \qquad (2)$$

Of course, knowledge of the relative error hinges on our ability to evaluate the functional on the fine mesh solution, $J(U_h)$. In [12] and [14] we circumvent this difficulty by approximating the output on the embedded mesh as a function of the coarse mesh flow and adjoint solutions.

$$J(U_h) \approx J(U_h^H) - \underbrace{(\psi_h^H)^{\mathrm{T}}\,\mathbf{R}(U_h^H)}_{\text{Adjoint Correction}} - \underbrace{(\psi_h - \psi_h^H)^{\mathrm{T}}\,\mathbf{R}(U_h^H)}_{\text{Remaining Error}} \qquad (3)$$

Where $\mathbf{R}(\,)$ is the spatial operator of the Euler solver, ψ, is the discrete adjoint solution and the notation $(\,)_h^H$ is used to indicate prolongation from the coarser to finer mesh.

References [12] [13] and [15] contain full details of the implementation, and demonstrate the order of convergence of both the adjoint correction and the remaining error terms. In the current work, the remaining error term in eq.(3) is computed by differencing the linear and quadratic prolongations of the adjoint solution. Earlier publications contain detailed verification exercises demonstrating that this formulation achieves its designed mesh-convergence rate for both the adjoint-correction and the remaining-error terms.[12,15]

3. Constant Error Database

Populating aerodynamic databases with simulation data requires hundreds/thousands of CFD simulations incurring significant processing costs. For this abstract, we've limited this expense by running only 2D cases in our databases. Upon acceptance, the two-dimensional examples contained in this abstract will be replaced with three-dimensional simulations, and the investigations and conclusions will be modified where appropriate.

As a baseline for investigating the role of discretization error in simulation-based aero-data, we start by computing an aero-database to a fixed error-tolerance. Our prototype is simply a Mach-α sweep over a NACA 0012 airfoil. In an effort to examine discretization-error and meshing requirements over a wide range of physics, our parametric space covers 12 Mach numbers from low subsonic to moderate supersonic at 10 different angles-of-attack. Figure 2 illustrates the extent of this Mach-α space with inset figures (shaded by local Mach number) to illustrate flow in various regimes.

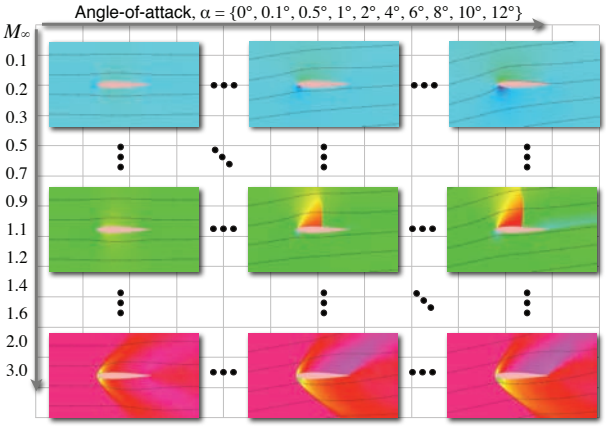


Figure 2: Mach-α wind-space covered by a prototypical aero-database showing examples of flow in various flight regimes. (12 Mach numbers) x (10 incidence angles) = 120 simulations in the database, Mach contours shown for selected cases.

For this study, the objective function was chosen to be a simple unweighted sum of lift and drag on the airfoil.

$$J(U_H) = C_l + C_d \tag{4}$$

To control discretization-error to a fixed value, every cases in the database was run adaptively until the estimate of the remaining-error term in eq.(3) was less than 0.008. With the functional in eq.(4), this tolerance translates to 80 counts of drag at zero-lift. This tolerance was purposely chosen relatively loose since we hope to be able to achieve it over a wide range of flow conditions. Figure 3 shows simulation results for this database through plots of lift, drag and moment versus alpha for each Mach number.

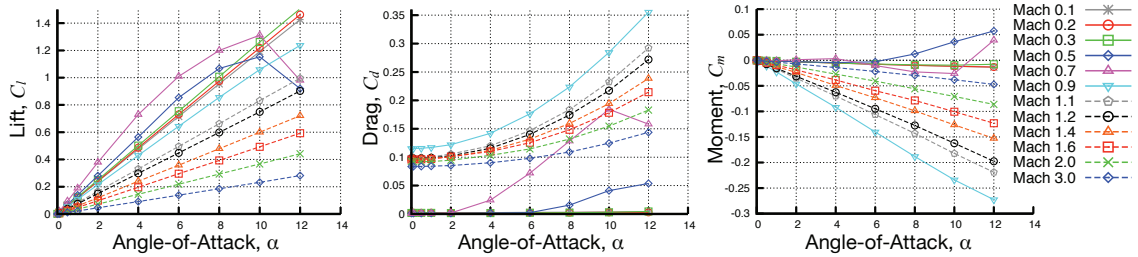


Figure 3: Lift, drag, and quarter-chord pitch-moment for aero-database in figure 2. All cases computed to constant-error tolerance of 0.008 on the approximation of the remaining-error in the functional $J = C_L + C_D$.

While figure 3 shows the primary aerodynamic data, figure 4 contains the central results for our current investigation of the error landscape. This figure shows quantitative measures of solution quality and cost. At the left of figure 4, we plot the magnitude of the remaining-error term in eq.(3) at each Mach number in the database as a function of angle-of-attack. The plot at the right shows the number of cells required to achieve this error level also as a function of M_∞ and α.

In examining the data in figure 4, the shaded region in the error plot indicates that all but one case in the database met our error tolerance of 0.008 on the functional $J(U_H) = C_l + C_d$. Moreover, we see that

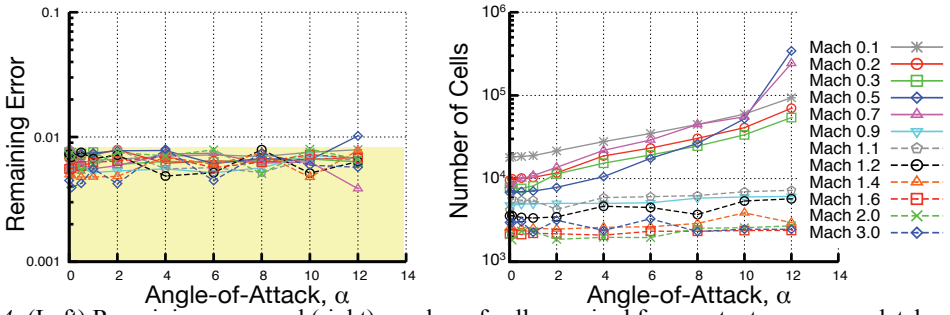


Figure 4: (Left) Remaining-error and (right) number of cells required for constant-error aero-database. Shaded region on left shows all but one simulation met requested error-tolerance of 0.008 on functional $J = C_L + C_D$.

the adaptive scheme controlled discretization-error so that the remaining error in most of the cases is clustered in a narrow band from 0.005 to 0.008. The narrowness of this band is a measure of control. In examining the cell-counts at the right of figure 4 we immediately note a clear division between the supersonic (dashed line) and subsonic (solid line) cases. Most noticeably, the supersonic cases generally achieved the desired level of error with over an order of magnitude fewer cells. In the subsonic regime, the resolution requirements increase with angle-of-attack and are Mach number dependent.

The division between subsonic and supersonic flow is a direct result of the domain of influence. In subsonic flow, error in any cell in the domain can potentially influence the functional. In supersonic flow, only cells in a rough diamond shaped zone formed by the bow shock in front and characteristics running upstream from the trailing edge in back can possibly affect the simulation (see figs. 9 & 24 in [14]). This dramatically limits the problem size in supersonic flow. Moreover, as the Mach number increases, this "refinement diamond" contracts in the cross-flow direction and the cell count drops with the area change (volume change in three-dimensions).

In the transonic regime, resolution requirements are driven by the functional's dependence on the precise location of the upper surface shock. As a result, the simulations at Mach 0.7 are among the most cell-intensive in the database. By Mach 0.9 the shocks are attached to the trailing-edge making these cases easy by comparison.

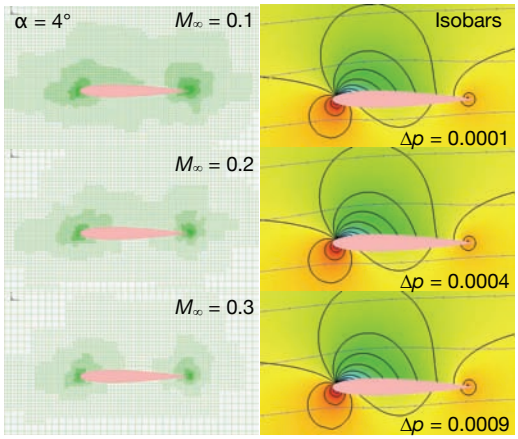


Figure 5: Final meshes and isobars of selected discrete solutions meeting constant error-tolerance on functional $J = C_L + C_D$ in near-incompressible flow.

While cell counts in the trans- and supersonic regimes are driven by flow physics, near the incompressible limit they are driven by our model of the governing equations. These simulations were preformed using the compressible form of the governing equations without any low-Mach preconditioning. Near the incompressible limit, the pressure coefficient becomes independent of Mach number giving rise to the well-known $1/M^2$ incompressible scaling. Figure 5 shows isobars in the discrete solution for the three lowest Mach numbers. Isobar levels at M_∞ of 0.1, 0.2 and 0.3 have been chosen to illustrate the approach to self-similarity in these nearly incompressible flows. Pressure signals at $M_\infty = 0.1$ are 9 times weaker than their counterparts at $M_\infty = 0.3$ in accordance with the inverse Mach-squared scaling. This represents nearly a 10 fold decrease in signal strength and therefore meeting the same error-tolerance requires substantially higher mesh resolution. This stiffness is simply an artifact of solving the unpreconditioned governing equations, and similar issues arise very near unit freestream Mach numbers.

4. Constant Mesh Database

Figures 6 shows a view of the computational mesh used in a database computed on a fixed computational mesh with ~7000 control volumes. This *fixed-mesh* database used the same computational mesh for all 120 simulations in the database. This represents the standard approach for constructing aerody-

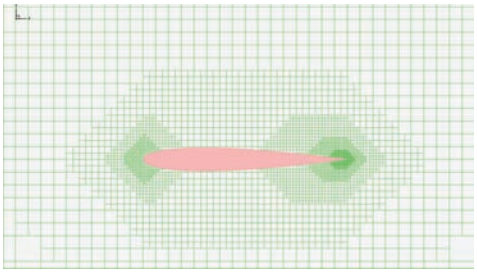


Figure 6: Computational mesh with ~7000 cells used for all cases in the *fixed-mesh* database.

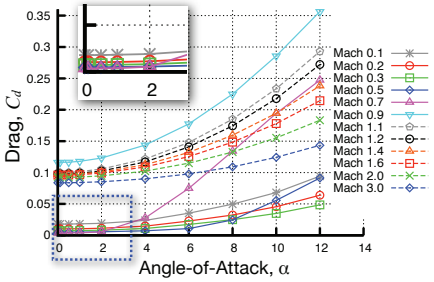


Figure 7: Drag in database computed on *fixed-mesh*. Inset shows poor prediction of drag at low Mach near α = 0.

namic databases today. Figure 7 shows the prediction of drag across the entire database as a function of angle-of-attack for all Mach numbers. In comparing these results with those in Fig. 3 we notice immediately the poor prediction in the subsonic regime at low incidence angles. Given the earlier discussion of flow sensitivity in this regime, these results are hardly surprising, but the quantification is illustrative. At low Mach numbers, inviscid flow theory predicts zero drag for this airfoil at zero lift. Instead, Fig.7 shows finite drag at zero lift, with results nicely sorted with the inverse of Mach number as we approach the incompressible limit.

The zero-lift drag prediction only hints at the corruption of this data due to discretization-error. For a fuller understanding, adjoint-based error-estimates were computed for all simulations using the methods outlined earlier, but with mesh adaptation turned off. Figure 8 shows the error remaining in the output functional across the fixed-mesh data for all Mach numbers as a function of incidence angle. Error in the database ranges from 0.002 to 0.08 – approximately 1.5 orders of magnitude. Not surprisingly, these data correlate quite closely with the resolution requirements shown at the right of figure 4. The failure of this mesh to meet these resolution requirements manifests itself as simulation error.

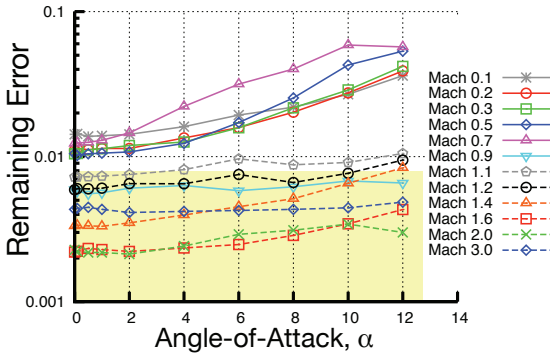


Figure 8: Remaining error in fixed-mesh database. Shaded region shows cases achieving error-tolerance of 0.008 on functional $J = C_L + C_D$.

The final paper will include a relative-error database and examples will focus on three-dimensional aero-data to ensure relevance to problems of practical engineering interest.

References

[1] TOP500 List Highlights, Nov. 2008. http://www.top500.org/lists/2008/11/highlights

[2] Biswas, R., Aftosmis, M.J., Kiris, C., and Shen, B.-W., "2007: Petascale Computing: Impact on Future NASA Missions." In Petascale Computing: Algorithms and Applications (D. Bader, ed.), Chapman and Hall /CRC Press, Dec. 2007.

[3] Mavriplis, D.J., Aftosmis, M.J., and Berger, M.J., "High resolution aerospace applications using the NASA Columbia supercomputer." *International Journal of High Performance Computing Applications*. **21**(1):106-126. Jan. 2007

[4] Pulliam, T.H., and Jespersen, D.C., "Large Scale Aerodynamic Calculation on Pleiades", (Abstract) 21st Internat. Conf. on Parallel CFD, Moffett Field CA., May 2009.

[5] Aftosmis, M.J., Berger, M.J., Melton, J.E., "Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry."*AIAA J.* **36**(6):952-960, Jun. 1998.

[6] Aftosmis, M.J., Berger, M.J., and Adomavicius, G., "A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries." *AIAA Paper 2000-0808*, Jan. 2000.

[7] Aftosmis, M.J., Berger, M.J, Biswas, R., Djomehri, M.J., Hood, R., Jin, H., and Kiris, C., "A Detailed Performance Characterization of Columbia using Aeronautics Benchmarks and Applications," *AIAA Paper 2006-0084*, Jan. 2006.

[8] Rogers, S. E, Aftosmis, M.J., Pandya, S.A., and Chaderjian, N.M, Tejnil, E., and Ahmad, J., "Automated CFD parameter studies on distributed parallel computers." *AIAA Paper 2003-4229*. 16th AIAA Computational Fluid Dynamics Conference, Jun. 2003.

[9] Chaderjian, N.M., Rogers, S.E., Aftosmis, M.J., Pandya, S.A., Ahmad, J.U., and Tejnil, E., "Automated CFD Database Generation for a 2nd Generation Glide-Back-Booster," *AIAA Paper 2003-3788*, June 2003.

[10] Murman, S.M., Aftosmis, M.J., and Nemec, M., "Automated parameter studies using a Cartesian method." *AIAA Paper 2004-5076*, Aug. 2004.

[11] Aftosmis, M.J., and Rogers S. E., "Effects of jet-interaction on pitch control of a launch abort vehicle," *AIAA Paper 2008-1281*, Jan. 2008.

[12] Nemec, M., and Aftosmis, M.J., "Adjoint error-estimation and adaptive refinement for embedded-boundary Cartesian meshes." *AIAA Paper 2007-4187*, 18th AIAA CFD Conference, Miami, FL., Jun. 2007.

[13] Nemec, M., and Aftosmis, M.J., "Adjoint Sensitivity Computations for an Embedded-Boundary Cartesian Mesh Method," Jol. Computational Physics (2007), doi:10.10.16/j.jcp.2007.11.018. http://dx.doi.org/10.1016/j.jcp.2007.11.018 (persistent link)

[14] Nemec, M., Aftosmis, M.J., and Wintzer M., "Adjoint-based adaptive mesh refinement for complex geometries," *AIAA Paper 2008-0725*, Jan. 2008.

[15] Nemec, M., and Aftosmis, M.J., "Adjoint Error Estimation and Adaptive Refinement for Embedded-Boundary Cartesian Meshes." *AIAA J.* to Appear, 2009.

[16] Venditti, D. A. and Darmofal, D. L., "Grid Adaptation for Functional Outputs: Application to Two-Dimensional Inviscid Flow," *Journal of Computational Physics* , Vol. 176, 2002, pp. 40–69.

A hybrid CPU/GPU parallel algorithm for coupled Eulerian and Vortex Particle Methods

Christopher P. Stone*, Earl P.N. Duque*, Christopher C. Hennes**
*Intelligent Light, East Rutherford, NJ 07070, USA
(Tel: 201-460-4700; e-mail: {cps,epd}@ilight.com)
**Vortex Consulting, Norman, OK 73072, USA
(e-mail: consulting@chrishennes.com)

Abstract: A combined CPU/GPU parallel algorithm for a hybrid Eulerian/Lagrangian CFD method is presented. Specifically, the optimized multi-GPU algorithm for direct Biot-Savart particle-particle interactions, associated with Vortex Particle Methods, is described in detail. The particle integration algorithm is implemented using multiple general purpose graphics processing units (GPU's) via NVIDIA's *Compute Unified Device Architecture* (CUDA). Details of the CUDA algorithm and the Eulerian/Lagrangian coupling method are described. Benchmark performance results for the Biot-Savart induction operation for velocity and velocity-gradient are presented.

Keywords: Vortex Particle Method, N-body problem, GPU, CUDA.

1. INTRODUCTION

The present algorithm couples an Eulerian CFD code, OVERFLOW-2, with the Vortex Particle Method (VPM). This coupled modeling method is intended for a rotorcraft simulation environment, specifically involving rotor-wake interactions. This type of simulation requires high fidelity modelling of the tip and wake vortex structures generated by the rotor blades and long term capture of these structures downstream. OVERFLOW-2 [1] is an unsteady, overset RANS algorithm that has been extensively employed for rotorcraft [2,3] and similar environments. However, as with all finite-difference methods, this algorithm is overly dissipative (i.e., not truly inviscid). Vortex Particle Methods (VPM) [4] solve the vorticity transport equation using discrete Lagrangian elements. Each of these elements contains a specific vectoral circulation or vortex strength. Their advection, numerically implemented as particle trajectories, replaces the non-linear terms encountered in discretized Eulerian methods. As such, they can convect vorticity without any intrinsic numerical dissipation making them well suited for wake simulations. Further, they are able to model sparse domains efficiently by only applying vortex elements were needed. To reduce the wake modelling cost and increase fidelity, Large Eddy Simulations (LES) methods can be applied within the VPM framework [5]. The combination of these two methods, OVERFLOW-2 in the near-field and VPM in the wake, is a logical compromise leveraging the other's strengths. Details of the coupling algorithm are briefly given in the following section. The primary focus of this paper is the performance acceleration for the VPM method using a hybrid CPU/GPU algorithm.

Central to Lagrangian vortex particle methods, the Biot-Savart induction operation is used to compute the velocity and velocity-gradient on individual particles. For a single particle, the discreet induction is a sum of the influences of all other particles. For a set of n non-singular particles with vectoral strengths ($\boldsymbol{\beta}$) and smoothing radii (σ), the induction for velocity at a discreet position ($\mathbf{x}$) can be written in pseudo-tensor form as:

$$u_i(x) = -\frac{1}{4\pi}\sum_l^N \left[K(R/\sigma)\ \frac{r_m\ \beta_n(\vec{y}_l)\ \epsilon_{imn}}{R^3} \right] \tag{1}$$

where $\mathbf{r} = (\mathbf{x} - \mathbf{y})$ is the particle separation vector, $R = |\mathbf{r}|$ is the particle separation distance, and $K(\gamma) = 1 - \exp(-\gamma^3)$ is the Gaussian smoothing function. The velocity-gradient tensor is obtained by direct differentiation of (1).

Evident from (1), the total number of particle-particle interactions for n particles is n^2 (i.e., quadratic). From symmetry, the number of interactions can be reduced by a factor of two for uniform or constant support radii (σ). A quick examination of the algorithm reveals n independent summations which can operate concurrently in parallel. That is, given n processing elements (PE), the Biot-Savart induction could operate in linear time, i.e., $O(n)$.

Common VPM simulations employ 10^5-10^7 particles; therefore, this is not realizable even for massive computational clusters due to size constraints and communication bottlenecks. In general, the direct summation method is not employed due to the quadratic complexity. Instead, most algorithms rely upon hierarchal tree methods [6], fast multipole methods (FMM) [7] or, most commonly, a hybridization [8] of the two with complexities of $O(n \log n)$ or better. (The complexity of FMM is formally $O(n)$; however, benefit is typically only realized for large n due to the high operation count per element.)

The alternative method implemented here employs commodity graphics processing units (GPU's) in parallel. Specifically, we employ NVIDIA's *Compute Unified Device Architecture* [9] (CUDA) multi-threaded design supporting their *Single Instruction Multiple Thread* (SIMT) paradigm. CUDA defines an extended instruction set within standard *C* to support SIMT. The CUDA-enabled product line from NVIDIA offers fine-grained data parallelism through a hierarchal multiprocessor and multi-core design. These devices are designed around one or more Streaming Multiprocessors (SM's), (e.g., 14 SM's for the GT 9800 and 30 for the Tesla C1060). Each SM in turn contains eight Scalar Processors (SP's) along with local on-chip shared-memory (currently 16 kilobytes). All PE's have access to the global memory at high bandwidth. Threads are scheduled rapidly via a hardware-based, zero-overhead thread workload manager on each SM.

Within CUDA, kernels are defined which operate independently upon an assigned data space. Parallelism is implicit to the kernels, i.e. each thread executes the kernel concurrently. To hide latency on the device, many thread kernels are launched and the hardware-based workload manager optimizes the throughput. The number of threads is often limited by available on-chip memory, not the workload. To take advantage of the processing and memory hierarchy of the device, threads are created in fixed-size *blocks*, e.g. 64 threads per block. Loads and stores from global memory are optimized for the thread block. That is, if each thread within a block issues a load for a successive array element simultaneously, the transaction is *coalesced* into a single per-block load instruction as opposed to multiple, repeated individual requests per-thread. (This does not imply the coalesced load *completes* in a single instruction.) Therefore, it is advantageous to keep the block threads synchronized when loads and stores from global memory is required. Synchronization between the threads within a specific block is achieved through a hardware instruction allowing efficient fine-grained parallelism.

As detailed previously, direct evaluation of (1) is very amendable to fine-grained parallelization available on CUDA-enabled GPU's. Examples of successful gravitational N-body problems on CUDA GPU's can be found in [12]. Details of the current Biot-Savart implementation in single and multiple GPU environments are presented in subsequent sections. Implications of GPU memory access patterns on throughput as well as scalability in a distributed heterogeneous environment are reported.

2. NUMERICAL ALGORITHM

The salient features of the parallel coupling OVERFLOW-2 with the VPM algorithm will be briefly outlined here. Details of OVERFLOW-2 can be found in the previously cited literature. Particles are seeded within the OVERFLOW-2 domain on a user-specified *seed* grid at points with positive (outward) vorticity flux. Particles advect through the OVERFLOW-2 domain using interpolation to determine velocity. These particles are labeled *passive*. Once particles exit the OVERFLOW-2 domain, they become *active* and both velocity and velocity-gradient are evaluated via (1). The induction on *active* particles includes both *passive* and *active*. In this manner, information from within the OVERFLOW-2 domain can be transmitted to the VPM domain in a manner similar to overset grid methods. Overlap of the two domains is used to maintain continuity.

The VPM component of the coupled algorithm for a single timestep is as follows:

a) Seed new vortex particles upon specified grid at locations with outward vorticity flux. Vorticity is computed through finite-difference on the seed grid using velocity at time t^n.

b) Redistribute the particle field to a uniform spacing to maintain algorithm accuracy using a 4^{th}-order Lagrangian interpolation function.

c) Identify particles as *passive* if residing within the OVERFLOW-2 grid domain. Otherwise, particles are labeled as *active*. A search algorithm similar to those used in overset grid methods locates the best available *donor* cell for a *passive* particle.

d) Sort the particle list packing all *passive* particles in an optimized order according to donor grid. *Active* particles are appending after all *passive* particles.

e) Integrate the particles through the specified timestep δt using either a two-stage Runge-Kutta or two-level Adams-Bashford method. *Active* particles directly compute velocity and velocity-gradient via (1). Velocity and vorticity strength for *passive* particles are acquired through interpolation from local donor cells with a tri-linear algorithm using velocity at time t^n.

f) Compute velocity on double fringe OVERFLOW-2 boundary points through (1) using all particles. Freestream (potential) flow is superimposed upon the induced velocity. This sets the updated velocity boundary condition using by OVERFLOW-2 to integrate from time t^n to $t = t^n + \delta t$.

In the final step shown above, pressure and density are currently assumed constant, which is non-rigorous. This is a difficulty with coupling incompressible VPM with compressible RANS algorithms. Work is underway to add a *full potential equation* (FPE) algorithm to the coupling process. Inclusion of the full velocity potential in the background field will allow compressibility effects and, therefore, a proper evaluation of boundary conditions for OVERFLOW.

2.1 Biot-Savart CUDA algorithm

The evaluation of (1) at a given location requires the position ($\mathbf{x}$) and vectoral strength ($\boldsymbol{\beta}$) for all particles. For non-singular methods the particle radius (σ) is required. Depending upon the application, σ can be assumed constant. On output, the induction returns the velocity and velocity-gradient. Therefore, the position, strength, and radius are sent to and the velocity/velocity-gradient retrieved from the GPU's main (global) memory via CUDA data transmission routines.

The current Biot-Savart CUDA implementation assigns one *target* thread per particle. The thread block size is variable from 32 to 128 with 64 generally yielding the optimal performance. On-chip memory limitations prevent larger block sizes. Multiple (100's to 1000's) thread blocks are spawned concurrently and scheduled by the manager. As noted, each thread operates on a unique *target* particle and initializes all elements of the solution data structure (containing velocity and velocity-gradient) to zero. Once initialized, the thread enters the main particle-interaction loop. A straightforward implementation loops over all particle positions one-by-one loading from global memory. The pseudo-code is shown below:

```
1   myIdx = blockIdx * blockDim + threadIdx;
2   Velocity[myIdx] = 0; Gradient[myIdx] = 0;
3   for k = 1, NumberOfSourceParticles
4       Velocity[myIdx] += F( Position[k], Strength[k], targetPosition[myIdx], σ );
5       Gradient[myIdx] += G( Position[k], Strength[k], targetPosition[myIdx], σ );
6   end
```

Code 1. Naive CUDA N-body algorithm.

Here, *myIdx* is a unique global index, a function of the block size (*blockDim*), block index (*blockIdx*) and local thread index (*threadIdx*). Note, the symbols *blockDim*, *blockIdx* and *threadIdx* are reserved in CUDA and are defined externally upon entry to the kernel. The arrays *Velocity*[], *Gradient*[], *Position*[] and *Strength*[] are hold velocity, velocity-gradient tensor, position and vectoral strength with implied data-types and assignments operators. *F*(...) and *G*(...) are functional implementations of (1) for a single particle-particle interaction. While algorithmically simple and correct, the above algorithm would suffer severely from memory latency during the repeated loads and stores from global memory. Further, CUDA does not cache global memory operations, compounding the limited global memory bandwidth. In total, each thread would issue, at a minimum, two global loads and two stores per iteration. Performance results using this and subsequent CUDA algorithms are presented later.

To optimize the algorithm, the stride in the above example is increased to match the thread block size (*blockDim*). At the beginning of each iteration, all threads concurrently load a coalesced set of particle positions and strengths from global memory to a pre-allocated shared-memory segment. As discussed earlier, on-chip shared-memory reads and writes are highly optimized (near register speed). Once all *sibling* threads (i.e., those in the same thread block) have loaded their assigned data, insured by a synchronization barrier instruction, all may proceed concurrently with the *blockDim* particle-particle interactions. Further, the position vector of the current thread index is loaded from global memory and velocity and velocity-gradient are declared locally to avoid non-local stores. This is shown in the following pseudo-code:

```
1   myIdx = blockIdx * blockDim + threadIdx;
2   myPos = targetPosition[myIdx]; myVel = 0; myUij = 0;
3   for k = 1, NumberOfSourceParticles, blockDim
4       sharedPosition[threadIdx] = Position[(k-1)*blockDim+threadIdx];
5       sharedStrength[threadIdx] = Strength[(k-1)*blockDim+threadIdx];
6       Synchronize Threads();
7       for n = 1, blockDim
8           myVel += F( sharedPosition[n], sharedStrength[n], myPos, σ );
9           myUij += G( sharedPosition[n], sharedStrength[n], myPos, σ );
10      end
11      Synchronize Threads();
12  end
```

Code 2. Tiled CUDA N-body algorithm.

By pre-loading a coalesced segment of global memory into shared-memory, the inner loop can proceed without any bottlenecks. Here, all data referenced by the kernel functions F(...) and G(...) resides in shared-memory and may proceed at peak throughput. This method, commonly referred to as *tiled* [10], has similarities to block matrix operations. In the *tiled* algorithm, each thread only issues $2n$ / *blockDim* loads in total; and, these loads will be at a much higher bandwidth since the concurrent request will be coalesced into a single fetch. Each thread pre-loads its target position to alleviate a non-aligned or random fetch further increasing the realized load bandwidth.

Finally, multiple CUDA GPU's are supported in this implementation by a simple divide and conquer method. Multiple parallel threads (*pthreads*) are assigned exclusive access to the available GPU's within a single compute node. For distributed environments, this same method is be employed by assigning one MPI process per node. This provides for a coarse-grained approached whereby the particle information must be explicitly transmitted between distinct memory systems. The current MPI implementation requires that the entire particle field be stored within each process, allowing the above algorithms to be easily extended to this environment.

3. PERFORMANCE

A brief set of experiments was conducted to assess the performance of the various CUDA algorithms. The tests were conducted on the *Lincoln* system at the *National Center for Supercomputing Applications* (NCSA). A subset of this system is designed for hybrid, multi-grained CPU/GPU algorithms: each node contains two NVIDIA Tesla C1060 GPU's and two quad-core (2.33 GHz) Intel E5345 CPU sockets. This multi-level design allows for a variety of distributed and shared-memory algorithms using both CPU and GPU methods. An instantaneous particle field result from a coupled OVERFLOW/VPM rotorcraft hover simulation will be used for all subsequent benchmarks. The particle field consists of 258,357 active and 232,920 passive particles (or, 258,357 targets with 491,277 sources).

A baseline experiment was first conducted using a parallel CPU-only algorithm. Here, (1) was implemented in parallel using OpenMP and run using all eight cores on a single node. This CPU-only version achieved 202 million particle-particle interactions per second (*ints*). Note, the number of particle-particle interactions in the present benchmark is 127 billion: requiring greater than 10 minutes for a single evaluation. This and all subsequent benchmarks have been tabulated in Table (1).

The *naive* algorithm was tested on one NVIDIA Tesla C1060 GPU. This resulted in a peak throughput of 114 million *ints*, only 56% of the CPU-only performance. As expected, direct loads and stores from GPU global memory results in poor performance. The *tiled* algorithm resulted in 5,976 million *ints*: a factor of 51.8x improvement over the naive code and 29.3x over the multi-threaded CPU version.

A final single-GPU experiment was conducted merging the two methods. Here, the *target* position, velocity, and velocity-gradient are locally declared but *source* position and vectoral strength are loaded directly from global memory. The resulting throughput decreased by only 1% compared to the original *tiled* method. The high performance resulting from this simplistic method can be attributed to the high computational intensity (CI) of the Biot-Savart induction operation and, secondly, to locally storing the target position and intermediate results during the induction loop.

Table 1. Biot-Savart benchmark results using multi-threaded (OpenMP) CPU, single GPU and multiple GPU algorithms. Throughput is given as million particle-particle interactions per second. Speed-up is relative to 8-core OpenMP CPU-only throughput.

Description	Throughput (million *ints*)	Speed-up
8-core OpenMP CPU version	202	N/A
1 GPU using *naive* code	114	0.56
1 GPU with *tiled* code	5,976	29.3
2 GPU's with 2 controlling threads	10,584	52.4
4 GPU's using MPI and pthreads	21,820	108
8 GPU's using MPI and pthreads	40,842	202

The scalability of the multiple GPU's has also been evaluated. As previously discussed, we allocate one MPI process per node and partition the available GPU's to multiple threads. For one, two and four nodes, the multi-level parallel method yielded 10.58, 21.82, and 40.84 *billion ints*, respectively, with linear scalability.

4. CONCLUSIONS

A hybrid CPU/GPU algorithm for direct Biot-Savart particle-particle interaction calculations within the Vortex Particle Method is presented. The algorithm employs multiple levels of parallelism to achieve near-optimal throughput. The GPU algorithm uses NVIDIA's *Single Instruction, Multiple Thread* paradigm implemented with CUDA to exploit fine-grained parallelism. A combined MPI and *pthread* CPU algorithm is utilized to manage distributed memory systems with multiple GPU's per node. Assuming the reported linear scalability is maintained, a 500,000 *source*/*target* particle simulation could be completed in less than 3 seconds per timestep with just 16 nodes. This high computational throughput offers a tractable cost when employing a direct summation method.

ACKNOWLEDGMENTS

The research was partially funded by the Government under Agreement No. W911W6-06-2-0008. The U.S. Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation thereon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. Computational support for this research was provided by TeraGrid under grants TG-ATM090028 and TG-MSS080014. TeraGrid resources were hosted by the *National Center for Supercomputing Applications* (NCSA).

REFERENCES

1. Buning, P.G., Parks, S.J., Chan, W.M., and Renze, K.J.. Application of the chimera overlapped grid scheme to simulation of space shuttle ascent flows. *Proceedings of the Fourth International Symposium on Computational Fluid Dynamics*, 1:132-137, 1991.
2. Lim, J.W. and Strawn, R.C.. Prediction of HART-II rotor BVI loading and wake system using CFD/CSD loose coupling. *AIAA Paper 2007-1281*, 2007.
3. Duque, E.P.N., van Dam, C.P., and Hughes, S.. Navier-stokes simulations of the NREL combined experiment Phase-II rotor. *AIAA Paper 99-0037*, 1999.
4. Cottet, G-H. and Koumoutsakos, P.. *Vortex Methods*. Cambridge University Press, 2000.
5. Gharakhani, A.. A Lagrangian vortex method for grid-free dynamic LES. *AIAA Paper 2005-4625*, 2005.
6. Barnes, J. and Hut, P.. A hierarchical $O(n \log n)$ force-calculation algorithm. *Nature*, 324:446-449, 1986.
7. Greengard, L. and Rokhlin, V.. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325-348, 1987.
8. Stock, M. and Gharakhani, A.. Toward efficient GPU-accelerated n-body simulations. *AIAA Paper 2008-608*, 2008.
9. NVIDIA. Programming guide. Technical report, NVIDIA, www.nvidia.com/cuda, 2008.
10. Nyland, L., Harris, M., and Prins, J.. Fast N-body simulation with CUDA. *GPU Gems 3*, pp. 676-695, 2008.

Numerical Drag Reduction Studies of Generic Truck Models Using Active Flow Control

Ramesh Agarwal*, Miles Bellman*, Jonathan Naber*

**Department of Mechanical, Aerospace and Structural Engineering*

Washington University in St. Louis, St. Louis, MO63131, USA

(Tel: 314-935-6091; email:rka@Wustl.edu)

Abstract: The effect of active flow control (AFC), using oscillatory blowing (synthetic jets) and steady suction/blowing on drag reduction of cars/trucks is numerically studied using the Unsteady Reynolds-Averaged Navier-Stokes (URANS) equations. Two generic models of trucks for which the experimental data are available are considered in the simulations. CFD simulations are in good agreement with the data. Both the experiments and simulations show that 10 to 15% reduction in drag can be achieved by active flow control using oscillatory blowing. A combination of oscillatory blowing and steady blowing or suction at appropriate locations on the rear face of the truck can further reduce the drag. These studies indicate that the AFC techniques can be employed to achieve substantial drag reduction.

1. INTRODUCTION

Almost all road vehicles – cars and trucks are bluff bodies. Therefore their aerodynamic drag is dominated by the pressure drag due to the flow separation at the rear end of the body. Furthermore, the wake of the body is unsteady, turbulent and three-dimensional due to continuous shedding of the vortices. The worldwide usage of ground vehicles is very extensive for transport of passengers and goods and it accounts for over 30% of CO_2 and other greenhouse gas (GHG) emissions. Therefore reducing the drag of these vehicles can make a major contribution to reduction in fuel consumption and GHG emissions. In recent years, a number of experimental and numerical studies have been conducted to explore the potential of drag reduction using AFC. Many of these studies have been conducted on Ahmed body [1] and other generic car/truck shapes. In this paper, we conduct numerical drag reduction study on a generic truck model used in the experiment of Seifert et al. [2] and on Ahmed body employed in the experimental and computational study (using Large Eddy Simulation model) of Wassen and Thiele [3].

2. CFD FLOW SOLVER

FLUENT is a commercially available numerical flow solver package, which is employed to compute the flowfields. This CFD software package solves the governing equations of incompressible, viscous fluid using a finite-volume method. It has several numerical algorithms for both steady and unsteady flow calculations on structured as well as unstructured grids. The software also has several zero-, one-, and two-equation turbulence models. "GAMBIT" is the pre-processing grid generation software that is provided with the FLUENT package; it is used to create the geometry as well as to generate the appropriate structured or unstructured meshes. In our calculations, we employ structured adaptive mesh on 2-D models and solve the URANS equations in conjunction with a two equation realizable k-ε model. The second-order upwind solver in FLUENT 6.2.16 is employed for the solution of the momentum equations. Pressure-velocity coupling in incompressible flow is solved using the Pressure-Implicit with Splitting of Operators (PISO) scheme, and the pressure is computed using the standard Poisson solver.

3. COMPUTED RESULTS

3.1 Computed Flow Field and Drag for a Generic Truck Model [2] without and with AFC

We consider the flow past a generic truck model used in the experiments of Seifert et al. [2].The geometry of the model is shown in Figure 1. Figure 2 shows the original mesh and Figure 3 shows the final adaptive mesh.

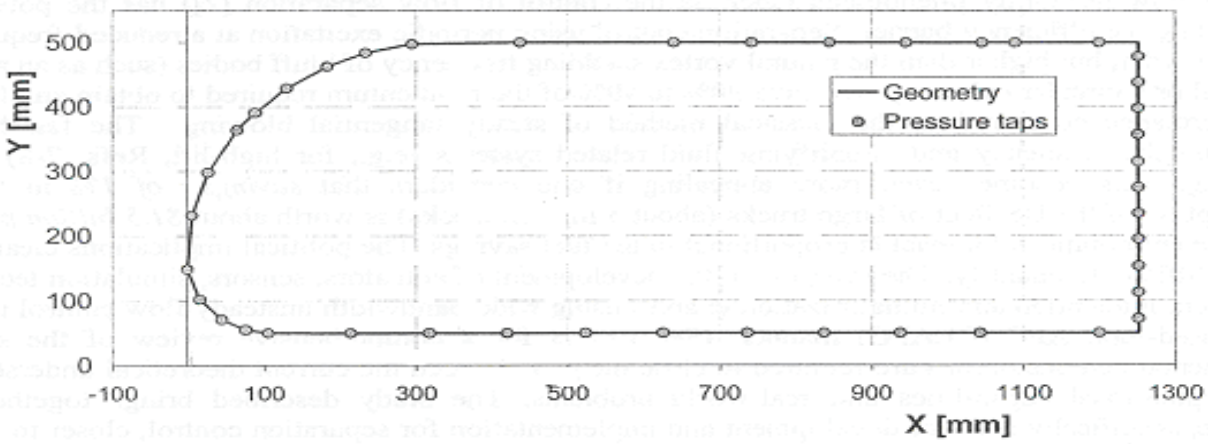


Figure 1: Generic Truck Model [2]

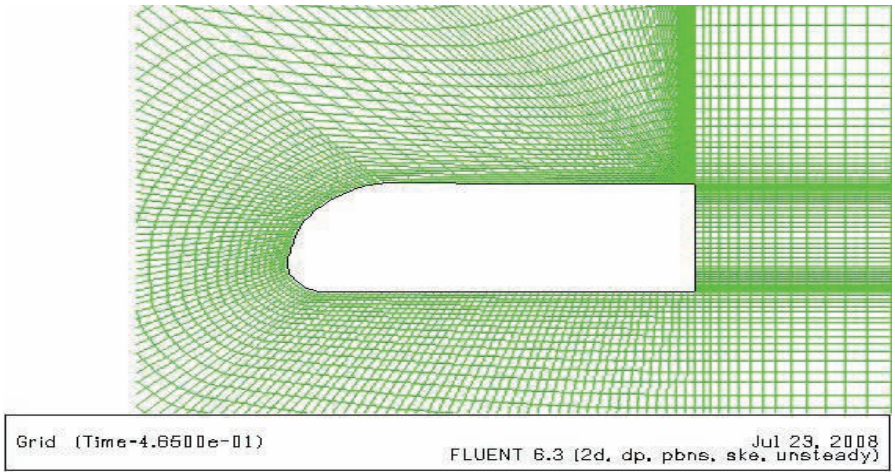


Figure 2: The mesh before adaptation

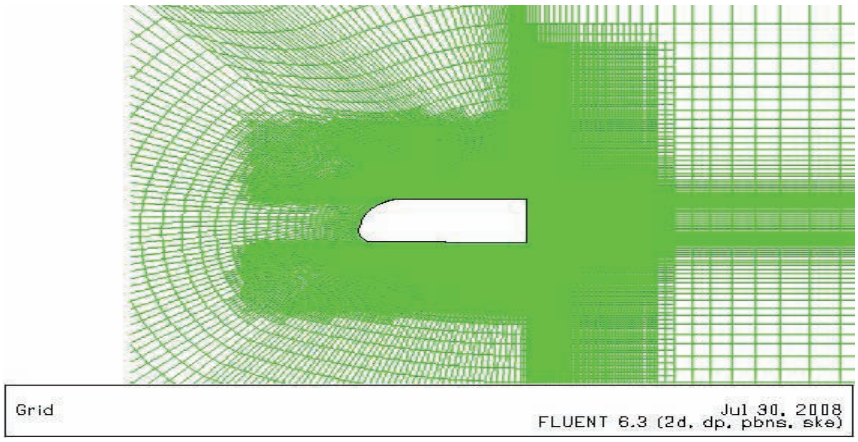


Figure 3: Final mesh after adaptation

Computations were performed at free-stream velocities of 10 m/s, 20 m/s, and 30 m/s used in the experiments of Seifert et al. [2]. The optimal adaptive mesh shown in Figure 3 with a second-order solver and k-ε model gave the best results within ±5% of the experimental values at all the three Reynolds numbers ($3.08x10^5$, $6.1x10^5$ and $9.24x10^5$ corresponding to free-stream velocities of 10 m/s, 20m/s and 30 m/s respectively). Figure 4 shows the comparison between the experimental and computed values of drag coefficient for three Reynolds numbers. The error bars on the graph are within the ±5% range. It is important to note that the computed drag coefficient decreases as the Reynolds number increases as expected from physical considerations; however this trend is not observed in the experimental data and has not been possible to diagnose.

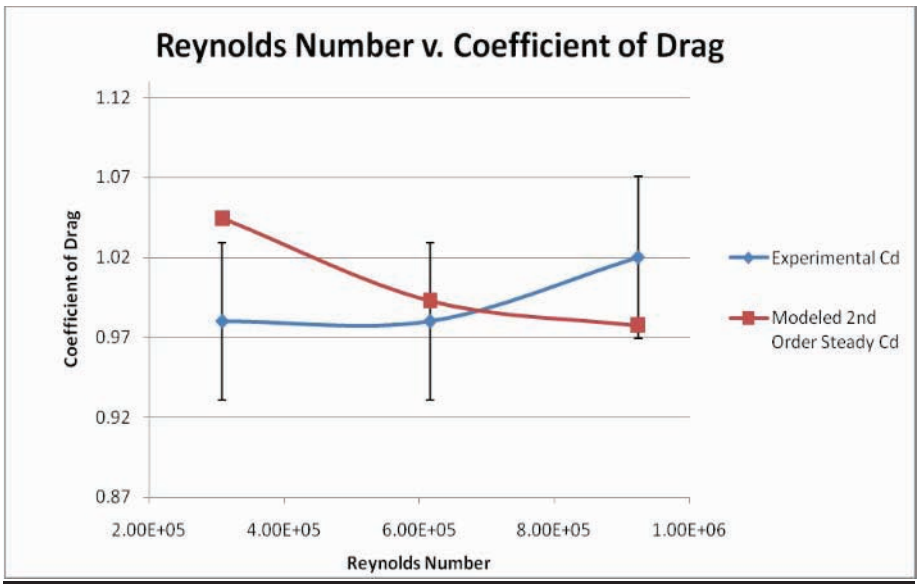


Figure 4: Variation of drag coefficient with Reynolds number; Blue: Experimental data, Red: Computations

Figure 5 shows the velocity contours in the flow field at a free-stream velocity of 10m/s or a Reynolds number Re = $3.08x10^5$.

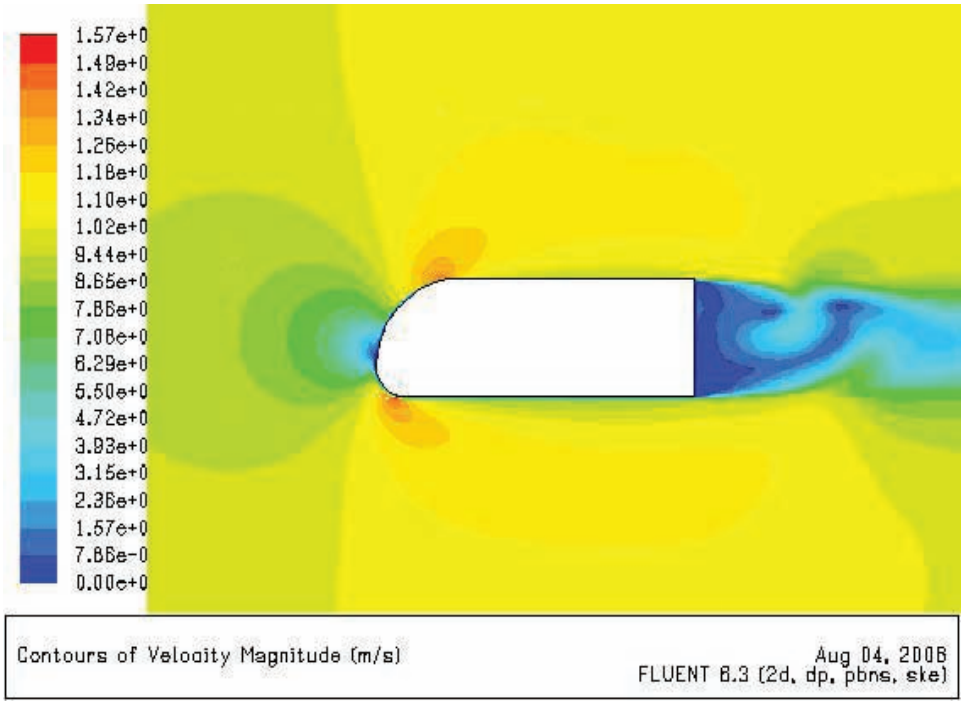


Figure 5: Velocity contours for flow past the truck at a free-stream velocity of 10 m/s; Re = $3.08x10^5$

After calculating the drag coefficient for three Reynolds numbers as shown in Figure 4, computations were performed with synthetic jets placed near the upper and lower corner at the rear of the truck. The amplitude of the synthetic jet velocity was taken to be half of the free-stream velocity. The frequency was 100Hz and the jet width was 1.7mm. As shown in Table 1, 8.94%, 13.41% and 15.62% reduction in drag (compared to baseline shown in Figure 4) was calculated for free-stream velocities of 10m/s, 20m/s and 30m/s respectively. These reductions represent major decrease in the aerodynamic drag of the generic truck model due to AFC. Figure 6 shows the variation of drag coefficient with Reynolds number without and with AFC. The results presented in this section clearly demonstrate the effectiveness of active flow control using synthetic jets in reducing drag of a truck.

Table1: Computed drag coefficient (C_d) without and with AFC; A = Jet amplitude, f = frequency, b = jet width U = free-stream velocity

Free-stream Velocity (m/s)	Reynolds Number	Experimental value of C_d	Computed C_d without AFC	Computed C_d with AFC A=.5*U, f=100Hz, b = 1.7mm	% Change in C_d with AFC
10	3.08E+05	0.98	1.04464442	0.9512261	-8.942595032
20	6.16E+05	0.98	0.99310098	0.859955849	-13.40700832
30	9.24E+05	1.02	0.977520171	0.824850755	-15.6180323

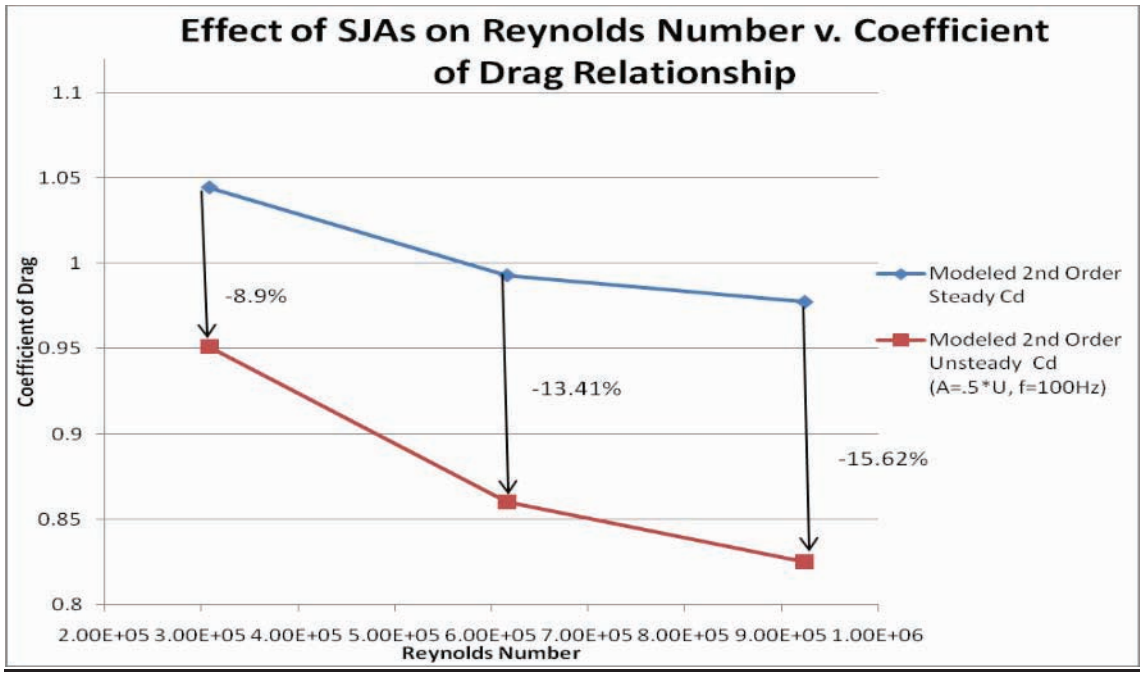


Figure 6: Variation of drag coefficient with Reynolds number without and with flow control using synthetic jets

3.2 Computed Flow Field and Drag of the 3D Ahmed Body [1, 3] without and with AFC

In Reference 3, both experimental and computational drag reduction studies on the 3D Ahmed car body, shown in Figure 7, were performed using steady blowing. Computational studies were performed using the Large Eddy Simulation model. A 6.4% reduction in drag coefficient was computed using the steady blowing from the upper

slanted surface. The baseline C_{d0} was 0.374 and the C_d with steady blowing was 0.350 for a free stream velocity of 7m/s. We have performed 3D computations using URANS equations with a k-ε model. We have computed the C_{d0} without AFC as 0.370 and C_d with steady blowing as 0.348. Our results with URANS are in excellent agreement with LES results reported in Reference 3. Computations have also been performed with a synthetic jet placed on the slant surface and another placed at the bottom corner. These computations show a further reduction in drag with a C_d of 0.322.

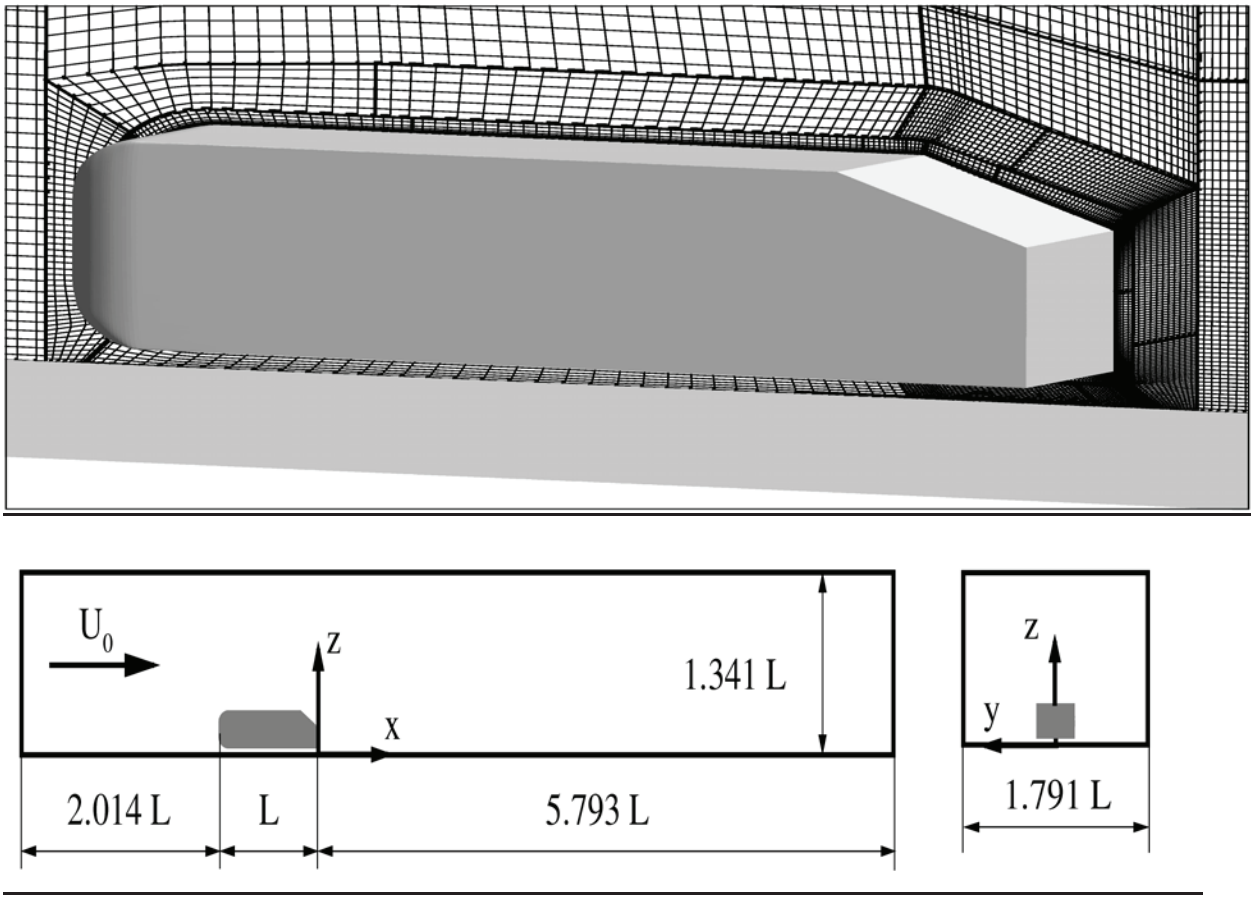


Figure 7: Geometry and dimensions of the Ahmed body [3]

4. CONCLUSIONS

The computational results presented in this paper clearly demonstrate that a significant reduction in drag (10 to 15 %) of truck like bodies can be achieved using active flow control with steady and/or oscillatory blowing. Computed results agree with the experimental data within 5%.

REFERENCES

1. Ahmed, S.R., Ramm, G., and Faltin, G. (1984). Some Salient Features of the Time-Averaged Ground Vehicle Wake, *SAE Technical Paper Series No. 840300.*

2. Seifert, A., Stalnov, O., Sperber, D., Arwatz, G., Palei, V., David, S., Dayan, I., and Fono, I. (2008). Large Trucks Drag Reduction Using Active Flow Control, *AIAA Paper 2008*.

3. Wassen, E. and Thiele, F. (2008). Drag Reduction for a Generic Car Model Using Steady Blowing, *AIAA Paper 2008-3771.*

Flow Modeling of Projectile Using Overset Flow Solver

Erdal Yilmaz and Shahrouz Aliabadi

Northrop Grumman Center for High Performance Computing
of Ship Systems Engineering,
School of Engineering and Technology,
Jackson State University, Jackson,
1230 Raymond Rd. MS 39204, USA
{erdal.yilmaz, saliabadi}@jsums.edu

Abstract: This study discusses details regarding our implementation of an overset mesh package into our parallel compressible flow code, CaMEL Aero for complex and moving body problems. The flow code is based on cell-centered finite volume formulation of the conservative Navier-Stokes equations using hybrid unstructured mesh topology with Detached Eddy Simulation for the turbulence modeling. We implemented SUGGAR/DiRTlib package for the overset mesh processing and operations. We tested our implementation with flow problem around a projectile. Parallel performance of our flow solver with overset on our parallel clusters is presented. Comparison with available experimental results will also be demonstrated. We will evaluate overall performance of the parallel overset capability as well.

Keywords: overset mesh, finite volume, parallel computing, compressible flow.

INTRODUCTION

Overset grid methodology is one of the highly demanded technologies for the numerical solution of the flows involving complex geometries and moving bodies. As the flow problems gets complicated more sophisticated approaches are required to handle challenges regarding mesh processing and operations. Over the year several overset grid tools have been developed by various research groups: SUGGAR/DiRTlib[1,2], PUNDIT[3], OVERFLOW-2DC[4,5], CHIMPS[6], BEGGAR[7], FASTRAN[8], Overture[9], and FVM[10]. All of these tools were developed to work with certain type of flow solvers therefore come with some deficiencies rather that being a general purpose overset tool. However, some of them are more versatile and practical than others. Most of the overset grid related tasks such as hole cutting, searching, blanking, communicate, and update etc are done with the minimum user-provided parameters hence reducing overall efforts to integrate it into different flow solvers. With all these nice practical features, however, one big challenge still lies ahead for highly complex and moving body flow problems on peta-scale parallel computing environment: whether scalability of the original parallel flow solver can be maintained.

In this study, we implemented Chimera overset capability, SUGGAR/DiRTlib package, into our compressible parallel flow code, CaMEL Aero [11,12], to solve flow problems involving moving objects or domains. The flow solver, CaMEL (Computation and Modeling Engineering Laboratory) Aero, is based on cell-centered finite volume formulation of the conservative Navier-Stokes equations using hybrid unstructured mesh topology with Detached Eddy Simulation (DES) for the turbulence modeling. The flow solver itself has been validated for various sets of complex flow problems in the past research studies. Suggar/DiRTlib provides an upper level library approach to tackle with the overset mesh implementation hence reducing most of the burden of overset mesh implementation. However, CFD developers still need significant time for the implementation of this overset tool without direct support from the developers of the tool package.

In the following sections, we will describe our overset implementation and an application to a compressible flow problem. Besides just implanting the overset tool into the flow solver, we need some pre- and post processing operations depending on the type of the mesh data format the solvers use since it has limited support for different mesh data formats. The problem we will solve for the demonstration of the capability is a projectile case at transonic to supersonic flow regimes with different roll rates and angles of attacks. We will compare our results against the experimental result [13] and other flow solutions. We will use static overset mesh methodology to solve this

problem then, we will introduce some oscillations to demonstrate dynamic overset mesh capability. Dynamic overset modeling involves unsteady simulations thus requires longer computational time due to cost of the overset mesh calculations at every time steps. Turbulence effect will be modeled using DES in all our simulations.

OVERSET IMPLEMENTATION INTO CAMEL AERO

In our overset grid implementation, SUGGAR handles overset grid assembly process and generates domain connectivity information (DCI file) that is needed for the exchange of the data among the overset mesh blocks. DCI file contains pairs of donor and fringe/receiver cells/nodes for the interpolation between mesh blocks, weights of interpolation for cells/node, blanked out cells and some other parameters specific to the grids. All mesh blocks are combined into one single block for the flow solver. However, SUGGAR can accept them separately and use a hierarchical block relationship. SUGGAR can run as a separate batch process or can be linked to the flow solver. For static overset, one without any mesh block moving, SUGGAR can be run only once at the beginning to generate DCI file. However, for moving body problems it should be run every time step to move the mesh accordingly as in the flow solver. It can run with multi-thread option to give better timing. DCi file generated by SUGGAR is used by DiRTlib. It as a library and therefore linked to the CaMEL flow solver. DiRTlib has several function calls to manage retrieving the DCI file, communicate between overlapping blocks, and update of solutions on the overlapped mesh elements. For static overset problems, DiRTlib implementation into CaMEL Aero is very clean. There is an initialization at the beginning and then flow solution update calls at each non-linear iteration for each time step. For mowing body problems overset related initialization should be done every time step.

In parallel case, DiRTlib needs block partition information additionally. Block partitioning is already in our flow solver generated by using ParMetis [14] library. Therefore, it is transferred to DiRTlib within the solver by a using DiRTlib function call. While DiRTlib uses the same parallel processes synchronously as the flow solver, in parallel run cases, SUGGAR runs as one process only. This would create bottleneck in the parallel efficiency, however it can be avoided if prescribed motion is applicable to the particular problem. In prescribed motion case, the domain connectivity information is a-priori computed by SUGGAR and saved in files for each time step in the simulation. Then, the flow solver simply loads this file for each time step. Though one can argue that this would hide the real cost affect of the overset mesh into parallel efficiency, it could also be perceived as mesh pre-processing. For moving body problems there will be more overhead to the total timing of the simulation.

TEST CASE AND RESULTS

We run a simulation for a projectile test case [13] using our parallel flow solver, CaMEL Aero, with the overset capability on one of our parallel clusters, one with 80 cores of Intel Xeon, 3.0 GHz, and 32GB memory per node. Initial speedup and efficiency results with the overset component up to 64 cores are presented in Figure 1. Note that timing cost for the overset part includes only DiRTlib contribution, not that of SUGGAR contribution for this particular case. DiRTlib time includes reading DCI file, all updates, and communication needed dues to overset mesh only. SUGGAR time includes time to generate domain connectivity file needed for solution update by DiRTlib. We will also study timing cost of running SUGGAR at different run modes as well as with different input parameters.

Current projectile solution involves two original mesh blocks: one surrounds the projectile and the other is bigger than projectile and covers projectile mesh. Current run is at Mach number 1.5 and zero degrees of angle of attack, AoA, without spinning. A coarse mesh with five million hybrid cells of hexahedra, prism, pyramid, and tetrahedron was generated. We will generate a much bigger mesh and do parallel performance study with that problem in our clusters. Validation of the overset implementation will include ranges of Mach number from supersonic Mach numbers up to 4.5 and AoA from 0 to 5 degrees. Figure 2 shows contours of pressure and mesh partitioning for 12 blocks. Timing breakdown of CaMEL Aero with overset module is given in Table 1. Note that overset time due to DiRTlib calls is constant, therefore percentage value increases as we increase number of processors.

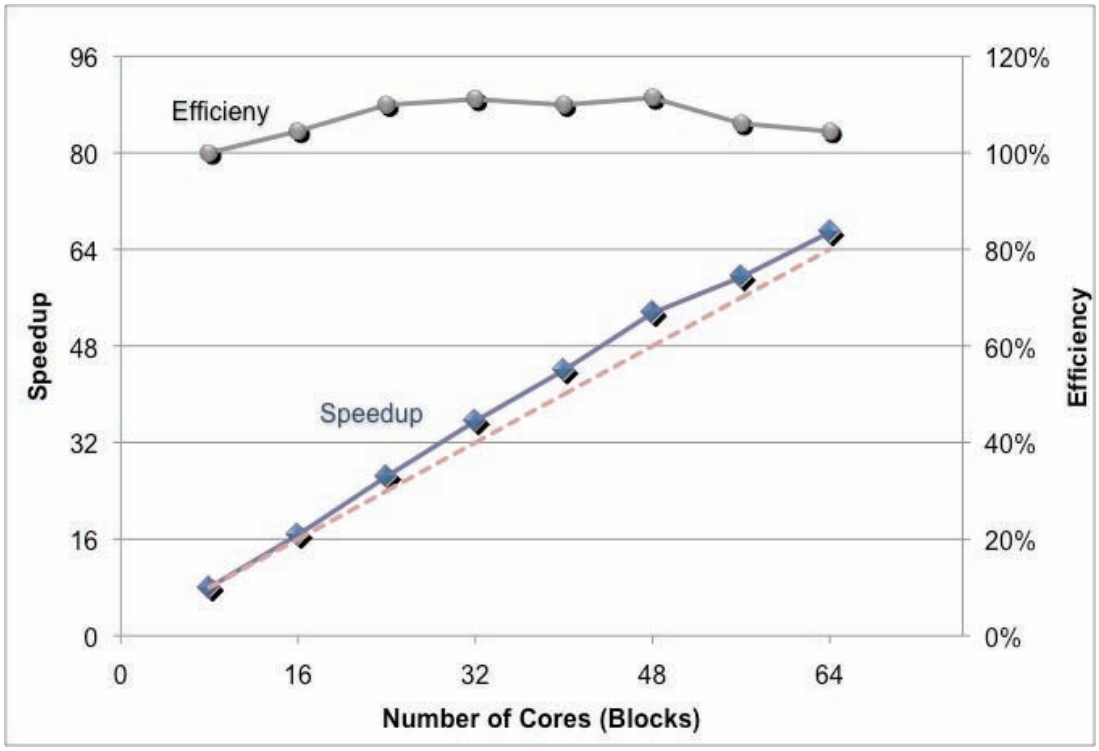


Figure 1: Parallel performance of CaMEL Aero with SUGGAR/DiRTlib for projectile case on the NGC Linux cluster. Flow solution is unsteady viscous with DES turbulence approximation and without moving. Timing includes DiRTlib time only. SUGGAR time is not included since the mesh is not moved in this case. Average timing breakdown: N-S Equations=68%, Turbulence=22%, Communication=6%, and Overset (DiRTlib only)=4%

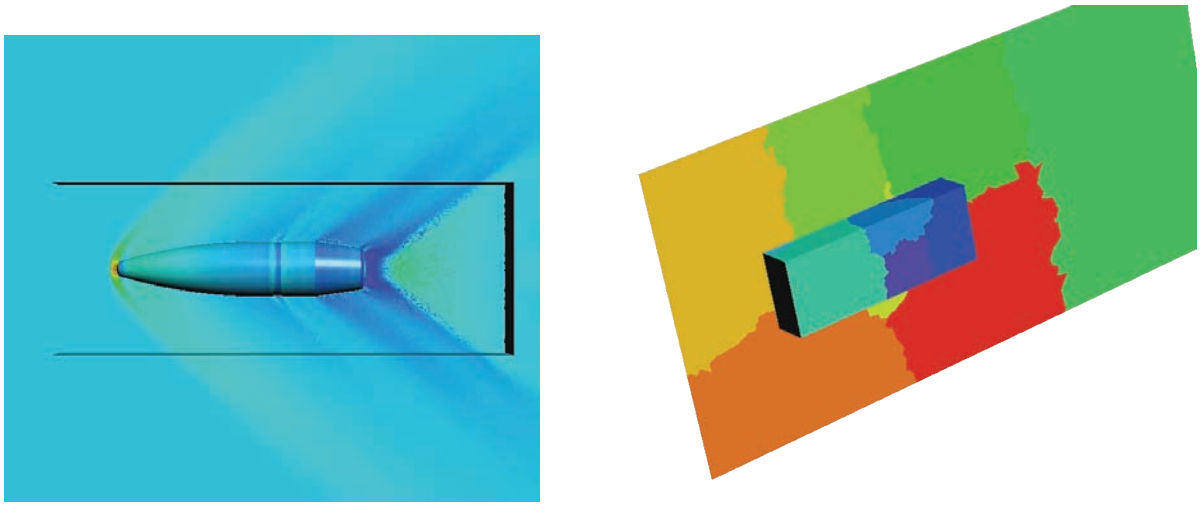


Figure 2: On the left: Presure contours of projectile test case at Mach 1.5 by CaMEL Aero code. Note that rectangular boundary denotes boundary of mesh blocks aroun the projectile. On the right: Partitioning of the combined mesh blocks for 12 processors showing continuationof the partitioning boundary through overset mesh boundary.

*Table 1: Timing breakdown of CaMEL Aero with overset module (DiRTlib only) for different number of processors. SUGGAR wall time took **67** seconds and run as separate once at the beginning of the time integration.*

# of cores	Total Time (sec/time step)	N-S solver (%)	Turbulence (%)	Communication (%)	Overset (%)
8	74	73	25	0.6	1.1
16	36	71	23	2.9	2.1
24	23	70	23	4.1	2.3
32	17	70	22	4.8	2.9
40	14	68	21	6.3	3.7
48	11	66	21	7.9	4.0
56	10	63	22	9.2	5.1
64	9	61	20	9.1	8.6

CONCLUSION

We have successfully implemented SUGGAR/DiRTlib overset mesh package into our CaMEL Aero compressible flow solver and demonstrated the applicability of it for flow problem around a projectile. This implementation took much less time than one can expect from developing the similar capability starting from scratch for this particular flow solver. However, we can conclude that more familiar with flow solver and overset methodology much shorter the implementation time. The overset function calls are kept at minimum so that less modification to the flow solver possible. The major bottleneck with this overset package appears as not be able to run SUGGAR in parallel the same way flow solver does. However, there are remedies to avoid this particular overhead depending on the nature of the problem and motion of the object. Parallel efficiency for this size of the problem is very promising as overset related updates and communications does not contribute significantly to the overall all timing cost of the problem.

ACKNOWLEDGEMENT

This work is funded in part by the Army Research Office under the auspices of the Department of the Army, Army Research Laboratory. Partial support for this publication made possible through support provided by Northrop Grumman Ship Building.

REFERENCES

1. Noack, R. W. (2003). Resolution Appropriate Overset Grid Assembly for Structured and Unstructured Grids. *16th AIAA Computational Fluid Dynamic Conference*. Orlando, FL. AIAA Paper 2003-4123

2. Noack, R.W. (2005). DiRTlib: A Library to Add an Overset Capability to Your Flow Solver. *17th AIAA Computational Fluid Dynamics Conference*. Toronto, Ontario, Canada. AIAA Paper 2005-5116.

3. Sitaraman, J., Floros, M., Wissink, A.M., and Potsdam, M. (2008). Parallel Unsteady Overset Mesh Methodology for a Multi-Solver Paradigm with Adaptive Cartesian Grids, *26th AIAA Applied Aerodynamics Conference. Honolulu, Hawaii*. AIAA Paper 2008-7177

4. Meakin, R. L. (2001). Object X-Rays for Cutting Holes in Composite Overset Structured Grids, AIAA Paper 2001-2537.

5. Buning, P. G., et. al. (2003). *OVERFLOW Users Manual*. NASA Langley Research Center.

6. Alons, J. et.al. (2006). CHIMPS: A High-Performance Scalable Module for Multi-Physics Simulations. *42nd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*. Sacramento, California.

7. Belk, D. M., and Maple, R. C., Automated Assembly of Structured Grids for Moving Body Problems, AIAA 95-1680, June 1995.

8. Wang, Z. J., Parthasarathy, V., and Hariharan, N. (1998). A Fully Automated Chimera Methodology for Multiple Moving Body Problems. AIAA Paper 98-0217.

9. Brown, D. L., Henshaw, W. D., and Quinlan, D. J. (1999). Overture: Object-Oriented Tools for Overset Grid Applications. AIAA Paper 99-3130.

10. Archambeau, F., Mechitoua, N., Sakiz, M. (2004) Code_Saturne: a Finite Volume Code for the Computation of Turbulent Incompressible Flows – Industrial Applications. *International Journal on Finite Volumes*. Vol. 1.

11. Tu, S., Aliabadi, S., Johnson, A., and Watts, M. (2005). A Robust Parallel Implicit Finite Volume Solver for High Speed Compressible Flows. *43rd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada. AIAA-2005-1396.

12. Aliabadi,S., Johnson, A., J. Abedi, J., Tu, S., and Tate, A. (2004). High Performance Simulation of Contaminant Dispersion on the Cray X1: Verification and Implementation. *Journal of Aerospace Engineering Computing, Information, and Communication,* (1542-9423) vol. 1 no. 8, pages (341-361).

13. Silton, S.I. (2002). Navier-Stokes Computations for a Spinning Projectile From Subsonic to Supersonic Speeds. *ARL-TR-2850*. U.S. Army Research Laboratory, Aberdeen Proving Ground, MD.

14. Schloegel, K. Karypis, G., and Kumar, V. (2002). Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*. Volume 14, Issue 3, pages 219 – 240.

PARALLEL ALGORITHMS/SOLVERS

Understanding the Performance of Hybrid MPI/OpenMP Programming Model for Implicit CFD Codes

Dinesh Kaushik[1], Satish Balay[1], David Keyes[2], and Barry Smith[1]

[1] Argonne National Laboratory, Argonne, IL 60439 USA,
{kaushik,balay,bsmith}@mcs.anl.gov
[2] Columbia University, New York, NY 10027 USA,
kd2112@columbia.edu

With the availability of large-scale multicore processor based clusters, the different software models for parallel programming require a fresh assessment. For physically distributed memory machines, the message passing interface (MPI) has been a natural and very successful software model. For another category of machines with distributed shared memory and nonuniform memory access, both MPI and OpenMP have been used with respectable parallel scalability. However, for clusters with several multicore processors on a single node, the hybrid programming model with threads within a node (OpenMP being a special case of threads because of the potential for highly efficient handling of the threads and memory by the compiler) and MPI among the nodes seems natural [1].

Two extremes of execution on hybrid architectures are often employed, due to their programming simplicity. At one extreme is the scenario in which the user explicitly manages the memory updates among different processes by making explicit calls to update the values in the ghost regions. This is typically done by using MPI, but can also be implemented with OpenMP. The advantage of this approach is good performance and excellent scalability since network transactions can be performed at large granularity. When the user explicitly manages the memory updates, OpenMP can potentially offer the benefit of lower communication latencies by avoiding some extraneous copies and synchronizations introduced by the MPI implementation. The other extreme is the case in which the system manages updates among different threads (or processes), e.g., the shared memory model with OpenMP. Here the term "system" refers to the hardware or the operating system, but most commonly a combination of the two. The advantages are the ease of programming, possibly lower communication overhead, and no unnecessary copies since ghost regions are never explicitly used. However, performance and scalability are open issues. For example, the user may have to employ a

technique like coloring to create nonoverlapping units of work to get reasonable performance. In the hybrid programming model, some updates are managed by the user (e.g., via MPI or OpenMP) and the rest by the system (e.g., via OpenMP).

In this paper, we evaluate the hybrid programming model using memory performance as a metric in the context of an unstructured implicit CFD code, PETSc-FUN3D [2]. The performance of many scientific computing codes is dependent on the performance of the memory subsystem, including the available memory bandwidth, memory latency, number and sizes of caches, etc. In addition, scheduling of memory transactions can also play a large role in the performance of a code. Ideally, the load/store instructions should be issued as early as possible. However, because of hardware (number of load/store units) or software (poor quality assembly code) limitations, these instructions may be issued significantly late, when it is not possible to cover their high latency, resulting in poor overall performance. OpenMP has the potential of better memory subsystem performance since it can schedule the threads for better cache locality or hide the latency of a cache miss. However, if memory bandwidth is the critical resource, extra threads may only compete with each other, actually degrading performance relative to one thread.

To achieve high performance, a parallel algorithm needs to effectively utilize the memory subsystem and minimize the communication volume and the number of network transactions. These issues gain further importance on modern architectures, where the peak CPU performance is increasing much more rapidly than the memory or network performance.

In a typical PDE computation, four basic groups of tasks can be identified, based on the criteria of arithmetic concurrency, communication patterns, and the ratio of operation complexity to data size within the task. These four distinct groups, present in most implicit codes, are vertex-based loops, edge-based loops, recurrences, and global reductions. Each of these groups of tasks stresses a different subsystem of contemporary high-performance computers. After tuning, linear algebraic recurrences run at close to the aggregate memory-bandwidth limit on performance, flux computation loops over edges are bounded either by memory bandwidth or instruction scheduling, and parallel efficiency is bounded primarily by slight load imbalances at synchronization points [2].

While implementing the hybrid model, the following three issues should be considered.

- Cache locality
- Work Division Among Threads

Table 1. Execution time on IBM BlueGene/P (four 850 MHz cores per node) for function evaluations only, comparing the performance of distributed memory (MPI alone) and hybrid (MPI/OpenMP) programming models.

	MPI Processes per Node			Threads per Node in Hybrid Mode		
Nodes	1	2	4	1	2	4
128	162	92	50	162	84	44
256	92	50	30	92	48	26
512	50	16	17	50	26	14

Table 2. Total number of linear iterations for the case in Table 1.

	MPI Processes per Node		
Nodes	1	2	4
128	1217	1358	1439
256	1358	1439	1706
512	1439	1706	1906

– Update Management

There are many implementations possible that strike a different balance of these factors. In Table 1, we present one such implementation for hybrid model where work is divided among threads in a manual way (as is done in the pure MPI case). Here, each MPI process calls MeTiS to further subdivide the work among threads, ghost region data is replicated for each thread, and "owner computes" rule is applied for every thread. We expect this implementation to give much better performance than when work is divided by the compiler. The performance data in Table 1 on up to 512 nodes (2048 cores) appear promising for the hybrid model. However, the performance advantage primarily stems from algorithmic reasons (see Table 2). It is well known in the domain decomposition literature that the convergence rate of single level additive Schwarz method (parallel preconditioner in PETSc-FUN3D code [2]) degrades with the number of subdomains. Therefore, the preconditioner is stronger in the hybrid case since it uses fewer subdomains as compared to pure MPI case. We believe this to be one of the most important advantages of the hybrid model. In our full paper, we will have more details on this issue, compare three different implementations of the hybrid model, and document performance data on more machines.

References

1. E. Lusk and A. Chan. Early experiments with the openmp/mpi hybrid programming model. In *Proceedings of the Fourth International Workshop on OpenMP (IWOMP 2008)*, pages 36–47, May 12–14, 2008. West Lafayette, Indiana.
2. W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. High performance parallel implicit cfd. *Journal of Parallel Computing*, 27:337–362, 2001.

Enabling temporal blocking for a lattice Boltzmann flow solver through multicore-aware wavefront parallelization

Johannes Habich, T. Zeiser, G. Hager, G. Wellein
Erlangen Regional Computing Center (RRZE), 91052 Erlangen, Germany
email: hpc@rrze.uni-erlangen.de

Abstract

This report presents first results of a pipelined wavefront parallelization approach applied to the lattice Boltzmann method in 3D. Threads executing successive time steps on a fixed spatial domain are scheduled to the same multicore processor chip having a shared cache. The efficient reuse of data in cache by successive threads implements a multicore-aware temporal blocking parallelization in a rather simple way.

Keywords: lattice Boltzmann, CFD, temporal blocking, multicore, wavefront

1 Introduction

Recently Datta et al. [1] have shown that substantial effort has to be put into optimizations for stencil based methods, such as the Jacobi iteration in 3D, to reach an optimal implementation on state-of-the-art multicore architectures. Spatial blocking techniques, as used in that work, divide the computational domain into more compact blocks and therefore can enhance locality making better use of the caches. Performance can often further be increased by doing multiple time steps on one block. This technique is called temporal blocking. With every additional time step performed, however, the block shrinks in each of the dimensions, leading to rather shorter loops and increased overhead which may severely limit the benefit of temporal blocking. Furthermore, specific platform-dependent tuning parameters, e.g. the block size in every dimension, have to be carefully adjusted. Hardware independent, so called cache oblivious algorithms, e.g. Frigo et. al [2], come at the cost of many data TLB misses as shown in [3] for a 3D lattice Boltzmann flow solver due to irregular access patterns.

This paper presents first results of a multicore-aware lattice Boltzmann method based solver in 3D using a pipelined wavefront parallelization approach. Owing to its explicit nature and a cellular automata like update rule, the basic lattice Boltzmann algorithm is easy to implement, optimize and analyze. Compared to, e.g. Jacobi solvers, the larger stencil, e.g. the D3Q19 stencil used here, require the use of even smaller block sizes for traditional temporal blocking techniques. To overcome this limitation a wavefront based temporal blocking technique is proposed, similar to the one described in [9] for an Jacobi solver. Parallelization of the algorithm does not partition the domain right away, but executes several threads on the same domain with a certain spatial displacement. The displacement must be large enough to sustain the computational correctness depending on the stencil used to avoid race conditions among the threads. All threads of a so-called wavesocket are bound to a single multicore chip with a shared cache, which reduces the access to memory to one load for the initial data and to one store at the end of the temporal iterations.

2 Multicore testbeds

The key architectural features of the three compute nodes evaluated in this report are presented in Tab. 1. Performance data was obtained for domains which exceed the cache size by far. Since stencil computations are data intensive, the attainable main memory bandwidths as measured with optimized stream benchmark [10] runs are shown as well. “Optimized” refers to the use of non-temporal stores which are important to get unbiased results on x86 architectures as described in [9].

Modern multicore processor have different numbers of processor cores, which can communicate in many different and sophisticated ways. To express which cache level is shared by how many cores, a convenient terminology is

		Woodcrest	Dunnington	Nehalem
Type		Xeon 5160 @3.0 GHz	Xeon 7460 @2.66 GHz	Xeon @2.66 GHz
L1 group	size [kB]	32	32	32
	TRIAD GB/s	3.7	3.0	11.6
L2 group	L2 size [MB]	4	3	0.25
	shared by # cores	2	2	1
	TRIAD GB/s	3.7	3.5	see L1
L3 group / socket	L3 size [MB]	-	16	8
	shared by # cores	-	6	4
	TRIAD GB/s	-	3.5	16.6
System	# sockets	2	4	2
	raw bw [GB/s]	21.3	34.0	51.2
	TRIAD GB/s	6.7	13.2	32.7

Table 1: Overview on cache group structure and STREAM TRIAD performance for the systems in the test-bed. Non-temporal stores were used throughout, so all bandwidth numbers denote actual bus traffic. STREAM array size was 20,000,000 elements.

introduced: The group of cores of a processor that share a certain cache is called a *cache group.* Thus, there are L1 groups (which consist of a single core on all current multicore designs), L2 groups, etc.. There can be multiple instances of any cache group on a multicore chip. Note that the focus lies on the cache levels available for data only, ignoring all instruction caches.

This report focuses on Intel based multi-socket platforms using state-of-the-art dualcore ("Woodcrest"), hexacore ("Dunnington") and quadcore ("Nehalem") variants. As the lattice Boltzmann kernel on a single core can already saturate most of the memory bandwidth on those systems, temporal blocking is the most promising method of optimization. For all tests the Intel Fortran compilers in version 11.0.069 were used.

3 Lattice Boltzmann method

The lattice Boltzmann method (LBM) has evolved over the last two decades and is today widely accepted in academia and industry for solving incompressible flow problems. Coming from a simplified gas-kinetic description, i.e. a velocity-discrete Boltzmann equation with appropriate collision term, it satisfies the Navier-Stokes equations in the macroscopic limit with second-order of accuracy [4,5]. Here the D3Q19 discretization model, i.e. 19 discrete velocities in three spatial dimensions, with the BGK collision operator is used.

The evolution of the single particle distribution function f_i is described by the following equation ($i = 0 \dots 18$):

$$f_i(\vec{x}+\vec{e}_i\Delta t, t+\Delta t) = f_i^{\text{coll}}(\vec{x},t) = f_i(\vec{x},t) - \frac{1}{\tau}\left[f_i(\vec{x},t) - f_i^{eq}\left(\rho(\vec{x},t),\vec{u}(\vec{x},t)\right)\right]$$

$$\text{with } f_i^{eq}\left(\rho(\vec{x},t),\vec{u}(\vec{x},t)\right) = \rho(\vec{x},t)w_i\left[1+\frac{3}{c^2}\vec{e}_i\cdot\vec{u}(\vec{x},t)+\frac{9}{2c^4}\left(\vec{e}_i\cdot\vec{u}(\vec{x},t)\right)^2-\frac{3}{2c^2}\vec{u}(\vec{x},t)\cdot\vec{u}(\vec{x},t)\right]$$

where f_i^{coll} denotes the "intermediate" state after collision but before propagation. ρ and $\vec{u}$ are the macroscopic quantities and are obtained as 0^{th} and 1^{st} order moments of f_i with regard to the discrete velocity $\vec{e}_i$, i.e. $\rho(\vec{x},t) = \sum_0^{18} f_i(\vec{x},t)$ and $\rho(\vec{x},t)\vec{u}(\vec{x},t) = \sum_0^{18}\vec{e}_i f_i(\vec{x},t)$. The Taylor-expanded version of the Maxwell-Boltzmann equilibrium distribution function [5,6] f_i^{eq} is given by Eq. 1, w_i are direction-dependent constants [6] and $c = \frac{\Delta x}{\Delta t}$ with the lattice spacing Δx and the lattice time step Δt. The equation of state of an ideal gas provides the pressure p, $p(\vec{x},t) = c_s^2\rho(\vec{x},t)$, with c_s as the speed of sound. The fluid's kinematic viscosity is determined by the dimensionless collision frequency $\frac{1}{\tau}$ according to $\nu = \frac{1}{6}(2\tau-1)\Delta x c$ with $\tau > 0.5$ due to stability reasons [4–6].

For solid wall boundaries, the boundary conditions are realized by the bounce-back rule [4,5], i.e. if a distribution is about to be propagated into a solid cell, the distribution function returns to the original cell but with reversed momentum. Bounce-back generally assumes that the wall is in the middle between the two cell centers. In the halfway formulation this leads to:

$$f_{\bar{i}}(\vec{x},t+\Delta t) = f_i^{\mathrm{coll}}(\vec{x},t)$$

with $\vec{e}_{\bar{i}} = -\vec{e}_i$ and $f_i^{\mathrm{coll}}(\vec{x},t)$ being the right hand side of Eq. 1. The fullway bounce-back on the other hand is given by:

$$f_{\bar{i}}(\vec{x},t+\Delta t) = f_i^{\mathrm{coll}}(\vec{x},t-\Delta t).$$

In order to apply the halfway bounce-back, the propagation is first done for fluid cells only and afterwards the distributions pointing to obstacles are handled by simply reversing the distribution in direction inside the fluid cell. In contrast, for the fullway bounce-back the fluid cells propagate the distributions into all surrounding cells. In the next time step, values stored at the obstacles' positions are reversed in direction and propagated back to the originating cells. This implementation uses the fullway bounce-back, as it can more easily be adapted to the pipelined parallel wavefront approach. Future work will also employ the halfway bounce-back.

3.1 Implementation

The basic LBM in 3D can be implemented with three nested loops traversing the computational domain and updating each lattice cell. As a starting point we use a mature and well optimized LBM kernel in 3D as described in [7, 13, 14]. In our discussion we use Fortran indexing, i.e. in multi-dimensional arrays the first/inner most index is consecutive in main memory, and focus on memory bandwidth bound problems.

A single LBM "sweep" through the complete domain is usually wrapped into an iteration loop, which performs several time steps. Often a two grid approach is implemented, i.e. one array holds the original data and one the updated data, allowing for simple implementation and parallelization. After each outer iteration, the grids are simply interchanged. Using the structure-of-arrays layout "SOA", F(i,j,k,Q,t), the overall data transfer on cache based architectures for a single lattice site update is $19*8*3$ Byte (for more details we refer to [7]). Together with the STREAM bandwidth numbers (cf. Tab. 1) one can easily estimate the maximum performance of the compute systems under consideration in terms of the fluid cell update rate given in million fluid cell updates per second (FluidMLUPs). The attainable STREAM bandwidth for the Woodcrest L2 cache group is measured as 3.7 GB/s (6.3 GB/s for the node respectively), therefore the upper limit is 8 FluidMLUPs (14 FluidMLUPs). For the Dunnington the limit is 7 FluidMLUPs for the L2 and L3 cache groups (28 FluidMLUPs for the node respectively), and the estimates for the Nehalem are 36 FluidMLUPs for the L3 cache group (71 FluidMLUPs for the node respectively). The code was parallelized by applying OpenMP `parallel do` work-sharing directives to the outermost (k) loop. ccNUMA data locality was ensured by parallelizing the initialization loops as well.

As can be seen from Fig. 2 for Woodcrest and Dunnington the results are in very good agreement with our estimations, both get 85 % of the attainable STREAM bandwidth. The Nehalem however performs only with 77 % of the estimated performance, which is due to the relatively short running inner (i) loop and the sub-optimal data layout, for this particular new hardware design. Benchmarks show that the F(i,Q,j,k,t) layout is able to get up to 85 % of performance as well. A detailed discussion however would be beyond the scope of this report.

Pipeline parallel wavefront implementation

The pipeline parallel wavefront approach does not perform several timesteps after another for a compact block of the domain. In contrast it traverses the whole computational domain assigned to one cache-group. For each cache group, one so-called wavesocket is launched. A wavesocket comprises as many independent threads as there are cores in the group. Note that it is essential to employ proper thread/core affinity to ensure that each wavesocket is always scheduled to the same cache group. In this preliminary report we focus on a wavesocket with two threads, i.e. two successive time steps are performed by the two threads, where the second update should be done on data available in the shared cache.

The basic idea is illustrated in Fig. 1 where *Thread 0* does the collision for all fluid cells in plane k and propagates the new distribution functions to three adjacent planes $(k-1,k,k+1)$ residing in the second array (see arrows down in Fig. 1).

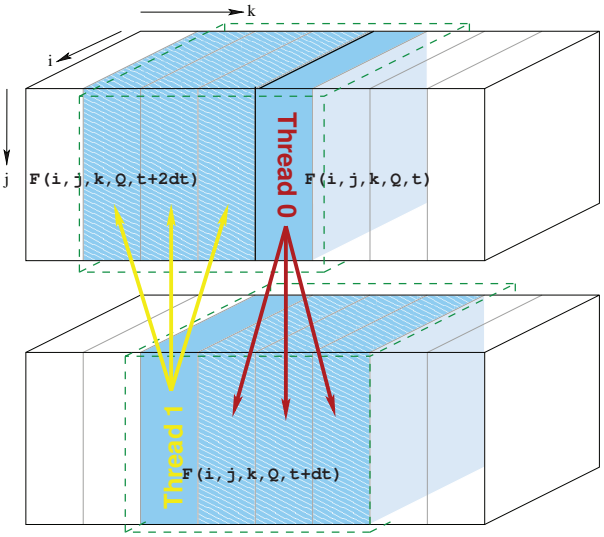


Figure 1: Time blocking through pipeline parallel processing by a two-thread wavefront group. Dashed boxes indicate (i, j) layers that must be kept in cache for optimal reuse of cache lines.

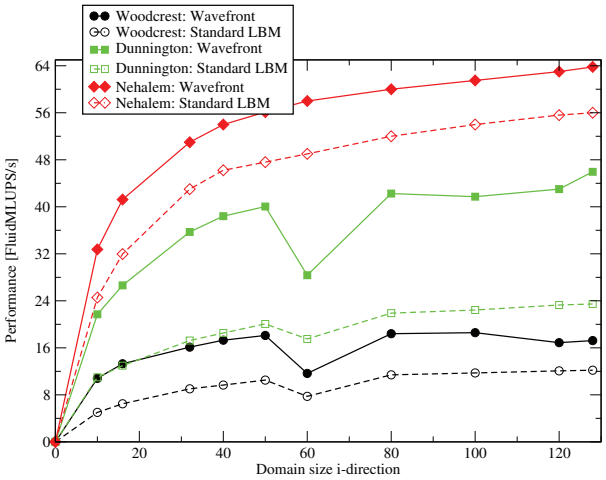


Figure 2: Performance comparison of the wavefront optimized LBM implementation with the standard implementation. All systems have been used with a maximum number of threads, i.e. 4 threads on Woodcrest, 16 threads on Nehalem (SMT on) and 24 threads on Dunnington. For the computational domain 600 was chosen in k-direction and 32 / 48 / 64 in j-direction for Woodcrest / Nehalem / Dunnington.

At the same time *Thread 1* can perform the collision of the updated $(t+dt)$ values in plane $k-2$ which do not have to be loaded from main memory again as long as the shared cache is large enough to hold the data in the lower dashed box of Fig. 1. Finally *Thread 1* propagates them (see arrows up in Fig. 1) back to the original array which then holds data for time step t ahead of the first wavefront (*Thread 0*) at larger k and the distribution functions for time step $t+2dt$ in the planes already visited by *Thread 0* before (at smaller k). It is obvious that in this approach the use of a second array spanning the complete computational domain is obsolete. It can be replaced by a temporary array holding 4 k planes, reducing the memory footprint of the LBM code by almost a factor of two.

If the shared cache is large enough to hold the two dashed boxes in Fig. 1, i.e. the complete temporary array and 4 k planes of the original array, then the overall data transfer for performing *two updates* on a single cell is just 19 load and 19 store operations, i.e. $2*19*8$ Byte per cell. This compares to $2*3*19*8$ Bytes per cell if both arrays are traversed twice as done in the original implementation. Thus, a maximum speed up of three may show up for this simple and straightforward wavefront implementation. Please note that the performance gain is larger than the number of time steps "blocked", because this implementation avoids the "Read for Ownership" (RFO) if writing back to the second array without reading it before. Here we store to the same array which has been loaded before and there is no need for the RFO if the cache is sufficiently large.

3.2 Results

In order to obtain accurate and comprehensive results, care has to be taken to properly pin the threads of common wavesockets to common cachegroups [12]. The results shown in Fig. 2 are a selection of various benchmark scenarios.

The wavefront parallelized version of the LBM leads to a 35 % higher performance on the Woodcrest node. Performance on the Nehalem system improves, however, only by 15 %. Due to the substantially better system balance of the Nehalem, the baseline performance is already high. The Dunnington, in contrast to the Nehalem, has a rather low system balance and is considered as a "bandwidth-starved" system design. However, it may be a somewhat prototypical for multicore chips to come. STREAM measurements show that already two of the six cores of a socket can saturate the full memory bandwidth. This leaves a lot of potential for temporal blocking techniques and so Dunnington shows the best gain from the pipeline parallel wavefront optimization, nearly doubling performance as compared to the standard implementation. For all systems, the performance gap is expected to grow with larger domain sizes.However, additional spatial blocking must be applied to a wavesocket's lattice partition to efficiently support

larger domains than those shown in Fig. 2. To avoid short inner loops blocking in j-direction is in order. It is also straightforward [9] to run more than two threads, e.g. 4 or 6 on the Dunnington system, in a single wavesocket further reducing the overall memory transfer. Furthermore, no cache optimizations for the kernel have been implemented so far. Work on all these topics is currently being done.

4 Conclusions and Acknowledgements

It was shown that pipelined wavefront parallelization efficiently implements temporal blocking for a large stencil based flow solver on Intel multicore CPUs. Especially highly bandwidth-starved systems show a substantial gain in terms of performance if this technique is applied.

We are indebted to Intel Germany for providing the "Nehalem" compute node through an early access program. This work was carried out within the Bavarian framework of KONWIHR. Financial support from BMBF through project SKALB (grant 01IH08003A) is gratefully acknowledged.

References

[1] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf and K. Yelick: *Stencil Computation Optimization and Auto-tuning on State-of-the-Art Multicore Architectures.* In: ACM/IEEE (Ed.): Proceedings of the ACM/IEEE SC 2008 Conference (Supercomputing Conference '08, Austin, TX, Nov 15–21, 2008).

[2] M. Frigo, C. E. Leiserson, H. Prokop and S. Ramachandran: *Cache-Oblivious Algorithms.* In: 40th Annual Symposium on Foundations of Computer Science, FOCS 99, Oct 17–18, 1999, New York, NY.

[3] T. Zeiser, G. Wellein, A. Nitsure, K. Iglberger, U. Rüde and G. Hager: *Introducing a parallel cache oblivious blocking approach for the lattice Boltzmann method.* Progress in CFD, Vol. 8, No. 1–4, pp. 179–188, 2008.

[4] S. Chen and G. D. Doolen: *Lattice Bolzmann method for fluid flows.* Annual Review of Fluid Mechanics Vol. 30 pp. 239–364, 1998.

[5] S. Succi: *The Lattice Boltzmann Equation - For Fluid Dynamics and Beyond.* Clarendon Press, 2001.

[6] X. He and L.-S. Luo: *Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation.* Phys. Rev. E, Vol. 56, pp. 6811-6817, 1997.

[7] G. Wellein, T. Zeiser, G. Hager, and S. Donath: *On the single processor performance of simple lattice Boltzmann kernels.* Computers & Fluids, Vol. 35, No. 8–9 pp. 910-919, 2006.

[8] D. J. Kerbyson and A. Hoisie: *Analysis of Wavefront Algorithms on Large-scale Two-level Heterogeneous Processing Systems.* In Proc. Workshop on Unique Chips and Systems (UCAS2), IEEE Int. Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, 2006.

[9] G. Wellein, G. Hager, T. Zeiser, M. Wittmann and H. Fehske: *Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization.* Submitted to IEEE International Computer Software and Applications Conference (COMPSAC 2009), Seattle, 2009

[10] J.D. McCalpin: *STREAM: Sustainable Memory Bandwidth in High Performance Computers.* `http://www.cs.virginia.edu/stream/`

[11] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin and J. C. Sancho: *A Performance Evaluation of the Nehalem Quad-Core Processor for Scientific Computing.* Parallel Processing Letters , Vol. 18 , No. 4, pp. 453–469 (2008).

[12] M. Meier: *Thread pinning by overloading pthread_create().* `http://www.mulder.franken.de/workstuff/pthread-overload.c`

[13] J. Habich: *Improving computational efficiency of lattice Boltzmann methods on complex geometries.* Bachelor's thesis, Chair of System Simulation, University of Erlangen-Nuremberg, Germany, 2006

[14] S. Donath, T. Zeiser, G. Hager, J. Habich and G. Wellein: *Optimizing performance of the lattice Boltzmann method for complex structures on cache-based architectures.* In: F. Huelsemann, M. Kowarschik, U. Ruede (Eds.): Frontiers in Simulation: Simulation Techniques - 18th Symposium in Erlangen, September 2005 (ASIM), pp. 728-735, SCS Publishing House, Erlangen, 2005.

On a parallel implementation of the BDDC method and its application to the Stokes problem

J. Šístek[1,4], **P. Burda**[3], **A. Damašek**[2], **J. Mandel**[4], **J. Novotný**[2,3], **B. Sousedík**[2,4]

[1]*Institute of Mathematics, Academy of Sciences of the Czech Republic, Prague, Czech Republic*
[2]*Institute of Thermomechanics, Academy of Sciences of the Czech Republic, Prague, Czech Republic*
[3]*Department of Mathematics, Czech Technical University in Prague, Czech Republic*
[4]*Department of Mathematical and Statistical Sciences, University of Colorado Denver, United States*

sistek@math.cas.cz, pavel.burda@fs.cvut.cz, damasek@it.cas.cz,
jan.mandel@ucdenver.edu, novotny@it.cas.cz, bedrich.sousedik@ucdenver.edu

keywords: BDDC, domain decomposition, iterative substructuring, Stokes flow

1. Introduction

Numerical solution of linear problems arising from discretization by finite elements is important in many areas of engineering. The matrix of the system is typically large, sparse, and often ill-conditioned. For large problems, iterative methods such as the preconditioned conjugate gradients (PCG) are usually less expensive in terms of memory and computational time. However, their convergence rate deteriorates with growing condition number of the solved linear system and good preconditioning becomes essential. The need of first-rate preconditioners tailored to the solved problem, which can be implemented in parallel, gave rise to the field of domain decomposition methods [12].

The Balancing Domain Decomposition based on Constraints (BDDC) [4, 9] is one of the most advanced preconditioners of this class derived for symmetric positive definite problems.

We have implemented a parallel version of the BDDC method and verified its performance on problems arising from linear elasticity problems. Then the method was successfully applied to the Stokes problem, which is beyond the standard theory of the BDDC method.

2. BDDC domain decomposition method

The Balancing Domain Decomposition by Constraints (BDDC) method can be understood as a preconditioner for large systems arising from finite element analysis. It was introduced by Dohrmann [4] in 2003 and the theory was developed by Mandel and Dohrmann in [8]. The preconditioner was reformulated by Li and Widlund in [7].

Let Ω be a bounded domain in $\mathbb{R}^2$ or $\mathbb{R}^3$, let U be a finite element space of piecewise polynomial functions v continuous on Ω and U' its dual space. Let $a(\cdot,\cdot)$ be a bilinear form on $U \times U$ and $f \in U'$, and let $\langle \cdot, \cdot \rangle$ denote the duality pairing of U' and U. Consider an abstract variational problem: *Find* $u \in U$ *such that*

$$a(u,v) = \langle f, v \rangle \quad \forall\, v \in U\,. \tag{1}$$

For the case of linear elasticity,

$$a(u,v) = \int_\Omega (\lambda(\nabla \cdot \mathbf{u}_h)(\nabla \cdot \mathbf{v}_h) + \frac{1}{2}\mu(\nabla \mathbf{u}_h + \nabla^T \mathbf{u}_h) : (\nabla \mathbf{v}_h + \nabla^T \mathbf{v}_h))\mathrm{d}\Omega, \tag{2}$$

$$\langle f, v \rangle = \int_\Omega \mathbf{f} \cdot \mathbf{v}_h \mathrm{d}\Omega. \tag{3}$$

Here solution $u = \mathbf{u}_h$ represents the discretized vector field of displacement, λ and μ represent the first and the second Lammé's constant, respectively, and **f** represents the external load.

For the case of steady Stokes flow we adopt the following slightly unusual notation

$$a(u,v) = \nu \int_\Omega \nabla \mathbf{u}_h : \nabla \mathbf{v}_h \mathrm{d}\Omega - \int_\Omega p_h \nabla \cdot \mathbf{v}_h \mathrm{d}\Omega + \int_\Omega \psi_h \nabla \cdot \mathbf{u}_h \mathrm{d}\Omega, \tag{4}$$

$$\langle f, v \rangle = \int_\Omega \mathbf{f} \cdot \mathbf{v}_h \mathrm{d}\Omega. \tag{5}$$

Solution $u = (\mathbf{u}_h, p_h)$ consists of the discretized vector field of velocity and the discretized scalar field of pressure, ν represents the kinematic viscosity of the fluid, $\mathbf{f}$ represents the external load, and $v = (\mathbf{v}_h, \psi_h)$.

For the case of linear elasticity, $a(u,v)$ is a symmetric positive definite bilinear form on $U \times U$, while for the Stokes problem, it is symmetric indefinite [2, 5].

Write the matrix problem corresponding to (1) as

$$Au = f. \tag{6}$$

The domain Ω is decomposed into N nonoverlapping subdomains Ω_i, $i = 1, ..., N$, with characteristic size H, which form a conforming triangulation of the domain Ω. Each subdomain is a union of several finite elements of the underlying mesh with characteristic mesh size h, i.e. nodes of the finite elements between subdomains coincide.

Unknowns common to at least two subdomains are called *boundary unknowns* and the union of all boundary unknowns is called the *interface* Γ.

Let W_i be the space of finite element functions on subdomain Ω_i and put

$$W = W_1 \times \cdots \times W_N. \tag{7}$$

It is the space where subdomains are completely disconnected, and functions on them independent of each other. Clearly, $U \subset W$.

The main idea of the BDDC preconditioner in the abstract form [10] is to construct an auxiliary finite dimensional space $\widetilde{W}$ such that

$$U \subset \widetilde{W} \subset W, \tag{8}$$

and extend the bilinear form $a(\cdot,\cdot)$ to a form $\widetilde{a}(\cdot,\cdot)$ defined on $\widetilde{W} \times \widetilde{W}$, such that solving the variational problem (1) with $\widetilde{a}(\cdot,\cdot)$ in place of $a(\cdot,\cdot)$ is cheaper and can be split into independent computations performed in parallel. Then the solution restricted to U is used for the preconditioning of (6). Space $\widetilde{W}$ contains functions continuous at selected coarse degrees of freedom such as values at selected nodes called *corners*. This space corresponds to a fictious mesh with connections limited to corners, as illustrated in Figure 1.

In computation, the corresponding matrix denoted $\widetilde{A}$ is used. It is larger than the original matrix of the problem A, but it possesses a simpler structure suitable for direct solution methods. This is the reason why it can be used as a preconditioner.

The projection $E : \widetilde{W} \to U$ is realized as a weighted average of values from different subdomains at unknowns on the interface Γ, thus resulting in functions continuous across the interface.

Let $r \in U'$ be the residual in an iteration of an iterative method. The BDDC preconditioner $M_{BDDC} : U' \to U$ in the abstract form (see [10]) produces the preconditioned residual $v \in U$ as

$$M_{BDDC} : r \to v = Ew,$$

where $w \in \widetilde{W}$ is obtained as the solution to problem

$$w \in \widetilde{W} : \widetilde{a}(w,z) = (r, Ez) \quad \forall z \in \widetilde{W}, \tag{9}$$

or in terms of matrices as

$$v = E\widetilde{A}^{-1}E^T r. \tag{10}$$

3. Numerical results

Our parallel implementation of the BDDC preconditioner has been extensively tested on problems with symmetric positive definite matrices arising from linear elasticity (e.g. [11]). The current version is based on the multifrontal massively parallel sparse direct solver MUMPS [1], which is used for factorization of matrix $\widetilde{A}$ in (10). The parallelism is obtained through the parallel direct solver used for factorization of the matrix of preconditioner, in combination with parallel PCG method.

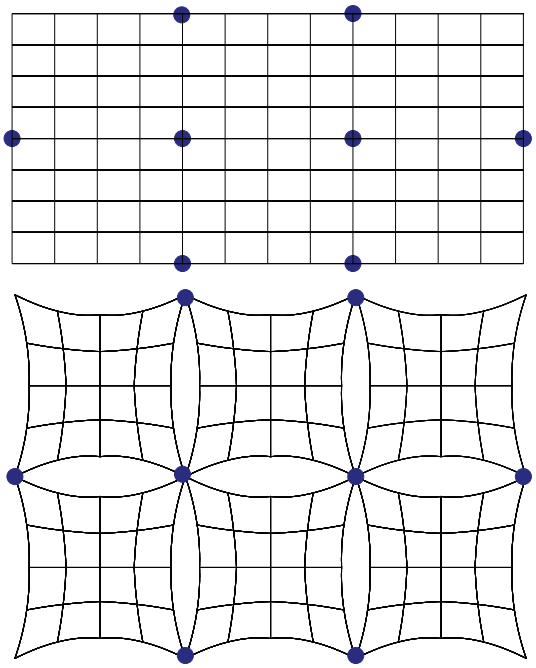


FIGURE 1. Example of an actual mesh (top) and the corresponding fictious mesh for construction of BDDC preconditioner (bottom), blue dots mark corners

The applicability of the preconditioner to the steady problem of Stokes flow was tested, and results are presented in this contribution. The system matrix of the Stokes problem is symmetric, but indefinite. For this reason, the standard theory of BDDC does not cover this case. A way to assure positive definiteness of the preconditioned operator based on BDDC was presented by Li and Widlund [6]. Their method relies on coarse degrees of freedom consisting of special averages on edges of subdomains. However, the approach is limited to piecewise constant pressure approximation. For P2/P1 and Q2/Q1 Taylor-Hood finite elements (e.g. [2]) used in our computations, we were not able to obtain contributive results by that method. Instead, presented problems were successfully solved with basic constraints as continuity at corners in the BDDC preconditioner setup. For the Stokes problem, matrix $\widetilde{A}$ is symmetric indefinite and as such is factorized by the MUMPS solver. Thus, the method leads to an indefinite preconditioner.

The method was first tested on the problem of lid driven cavity, a popular benchmark problem for methods for viscous flow. The domain is a unit square with homogeneous boundary conditions except horizontal velocity prescribed on the upper side. Thus, the entire motion in the cavity is driven by viscosity of the fluid. The case of uniform mesh of 128 $\times$128 Q2/Q1 elements was chosen. It was divided into 8 subdomains by METIS package (Figure 2).

Resulting streamlines and plot of pressure for Reynolds number $10,000$ are presented in Figure 3. Streamlines are symmetric along the vertical centreline for the Stokes problem.

Solution of the problem by our earlier solver based on a serial frontal algorithm took 231 seconds on one 1.5 GHz Intel Itanium 2 processor of SGI Altix 4700 computer in CTU Supercomputing Centre, Prague, compared to 17.2 seconds on 8 processors of the same computer necessary for the solution by the new implementation of BDDC. The stopping criterion of PCG was chosen as $\|r\|_2/\|g\|_2 < 10^{-3}$, resulting in 59 PCG iterations.

In the second example, a geometry with a sudden reduction of diameter is considered. Flow in this geometry described by the Navier-Stokes model was studied in [3], with respect to precise solution of corner singularities. Due to the symmetry of the channel, only the upper part is considered in the computation. Division into 4 subdomains obtained by METIS is presented in Figure 4.

Solution obtained by BDDC method at Reynolds number 250 for the Stokes flow is presented in Figure 5. The stopping criterion of PCG was chosen as $\|r\|_2/\|g\|_2 < 10^{-3}$, resulting again in 59 PCG iterations. Note, that fluid flows from right to left in the plot of pressure in order to show the situation at corners of domain.

To investigate the performance of the BDDC preconditioner in combination with standard iterative methods for general matrices, namely BICGSTAB and GMRES, we have also performed several preliminary experiments with our serial code written in MATLAB. In Table 1, we compare the resulting number of iterations of these methods

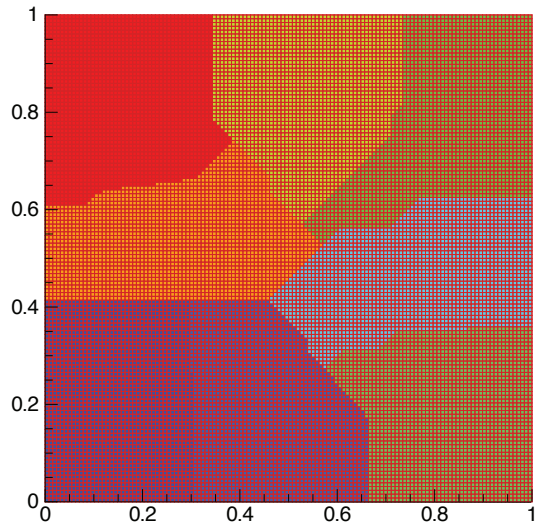


FIGURE 2. Mesh and its division into 8 subdomains for lid driven cavity, 128×128 elements

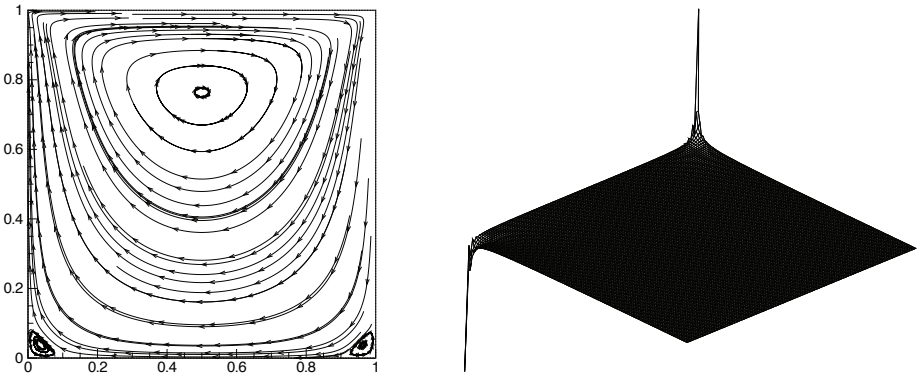


FIGURE 3. Streamlines (left) and pressure (right) for lid driven cavity, 128×128 elements

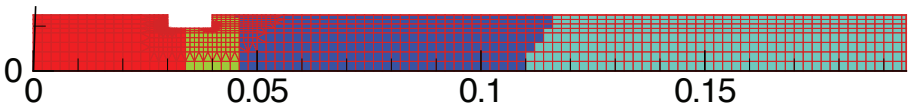


FIGURE 4. Mesh and its division into 4 subdomains for channel with sudden reduction of diameter, only the upper part of the channel is considered for symmetry.

preconditioned by BDDC and by the ILU preconditioner for several values of treshold τ for dropping entries in incomplete factorization for the cavity problem. The desired tolerance of relative residual for these methods was chosen as $\|r\|_2/\|g\|_2 < 10^{-8}$. Where 'n/a' is present in the table, BICGSTAB failed to converge.

iterative method	without preconditioner	BDDC corners only	BDDC corners+faces	ILU $\tau = 10^{-3}$	ILU $\tau = 10^{-4}$	ILU $\tau = 10^{-5}$
BICGSTAB	n/a	45	22	n/a	331	10
GMRES	759	49	38	472	87	18

TABLE 1. Number of iterations for BICGSTAB and GMRES without preconditioning, and preconditioned by BDDC and ILU, lid driven cavity.

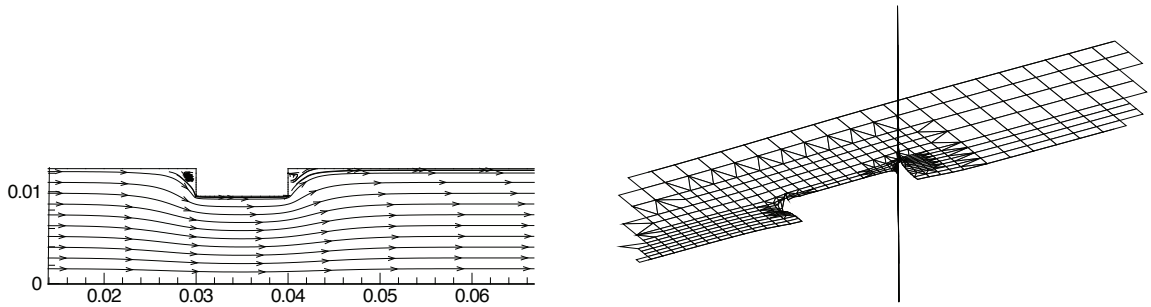


FIGURE 5. Detail of streamlines (left) and pressure (right) for channel with sudden reduction of diameter, Re = 250

4. CONCLUSION

In our contribution, we present a parallel implementation of the BDDC preconditioner. After a verification of the solver on a number of problems from linear elasticity analysis, we explore the application of BDDC to problems with indefinite matrices, namely the Stokes problem. Although the available theory either does not cover this case, or treats it differently [6], the presented experiments suggest promising ways for this effort. Without claiming that this is the general case, we have performed several experiments, for which PCG was successfully used even if the system was indefinite. The reason why a breakdown was not observed lies probably in the indefiniteness of the BDDC preconditioner for this case and deserves further investigation. Our serial experiments also led to promising results for combination of BDDC method with standard iterative methods for solving systems with general matrices, such as BICGSTAB and GMRES.

Acknowledgement: This research has been supported by National Science Foundation under grant DMS-0713876, by Czech Science Foundation under grant GA CR 106/08/0403, and by the Grant Agency of the Academy of Sciences of the Czech Republic under grant IAA200600801. A part of this work was done during Jakub Šístek's visit at the University of Colorado Denver.

REFERENCES

[1] AMESTOY, P. R., DUFF, I. S., AND L'EXCELLENT, J.-Y. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Engrg. 184* (2000), 501–520.

[2] BREZZI, F., AND FORTIN, M. *Mixed and hybrid finite element methods*, vol. 15 of *Springer Series in Computational Mathematics*. Springer-Verlag, New York, 1991.

[3] BURDA, P., NOVOTNÝ, J., AND ŠÍSTEK, J. Precise FEM solution of a corner singularity using an adjusted mesh. *Internat. J. Numer. Methods Fluids 47*, 10–11 (2005), 1285–1292.

[4] DOHRMANN, C. R. A preconditioner for substructuring based on constrained energy minimization. *SIAM J. Sci. Comput. 25*, 1 (2003), 246–258.

[5] ELMAN, H. C., SILVESTER, D. J., AND WATHEN, A. J. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 2005.

[6] LI, J., AND WIDLUND, O. B. BDDC algorithms for incompressible Stokes equations. *SIAM J. Numer. Anal. 44*, 6 (2006), 2432–2455 (electronic).

[7] LI, J., AND WIDLUND, O. B. FETI-DP, BDDC, and block Cholesky methods. *Internat. J. Numer. Methods Engrg. 66*, 2 (2006), 250–271.

[8] MANDEL, J., AND DOHRMANN, C. R. Convergence of a balancing domain decomposition by constraints and energy minimization. *Numer. Linear Algebra Appl. 10*, 7 (2003), 639–659.

[9] MANDEL, J., DOHRMANN, C. R., AND TEZAUR, R. An algebraic theory for primal and dual substructuring methods by constraints. *Appl. Numer. Math. 54*, 2 (2005), 167–193.

[10] MANDEL, J., AND SOUSEDÍK, B. BDDC and FETI-DP under minimalist assumptions. *Computing 81* (2007), 269–280.

[11] ŠÍSTEK, J., NOVOTNÝ, J., MANDEL, J., ČERTÍKOVÁ, M., AND BURDA, P. BDDC by a frontal solver and stress computation in a hip joint replacement. *Math. Comp. Simul.* (2009). Available online at http://dx.doi.org/10.1016/j.matcom.2009.01.002.

[12] TOSELLI, A., AND WIDLUND, O. *Domain decomposition methods—algorithms and theory*, vol. 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2005.

Parallel implementation of the adaptive Aitken-Schwarz method for non separable operator

T.Dufaud[1] and D.Tromeur-Dervout[1]

[1] Université de Lyon, Université Lyon 1, CDCSP/ICJ-UMR5208-U.Lyon1-ECL-INSA-CNRS,
43 blvd du 11 novembre 1918,F-69622 Villeurbanne-Cedex, France
{dtromeur,dufaud}@cdcsp.univ-lyon1.fr

Abstract: This paper focus on the parallel implementation and evaluation of the Aitken-Schwarz method when it applied to 3D Darcy flow where the permeability field is randomly distributed and exhibits strong variations. For this problem with a non-separable operator the matrix $\mathbb{P}$ involved in the acceleration is no more diagonal and must be built adaptively as in [2] for 2D problems. Special care is brought to the building of $\mathbb{P}$ with respect to the parallel architecture capabilities.

Keywords: Schwarz domain decomposition, Aitken's acceleration, adaptive computation, porous media

1 INTRODUCTION

The Aitken-Schwarz method [1, 4] demonstrated great efficiency on metacomputing framework for separable operators [3]. For non separable operators the decomposition in "Fourier" modes of the iterate solution at the artificial interface generated by the domain decomposition is no more available. Then the $\mathbb{P}$ matrix arising in the Aitken acceleration is no more block diagonal, and the Aitken formula cannot be applied for each "Fourier" mode separately. [2, 5] report an extension of the Aitken-Schwarz to non separable operators, where the matrix $\mathbb{P}$ is built adaptively with respect to the modes that have not converged. The present work focus on the parallel implementation and evaluation of this method when it applied to Darcy flow where the permeability field is randomly distributed and exhibits strong variations.

2 THE AITKEN-SCHWARZ METHOD APPLIED TO DARCY FLOW

2.1 The Darcy flow equation with strong variation in the permeability field

On the macroscopic scale, a porous medium can be described by a model where the solid and the fluid occupy the entire volume. The medium is regarded as a homogeneous domain and modeled as a continuum where a representative volume element is larger than the average pore size but much smaller than the length scale of the system. For this model of saturated flow in homogeneous porous media, the balance of momentum is given by Darcy's law:

$$\left\{ \begin{array}{lcl} \mu u + K\nabla p & = & 0, \text{ in } \Omega \\ \nabla . u & = & f, \text{ in } \Omega \end{array} \right. \tag{1}$$

$$p = 0 \text{ on } \partial\Omega. \tag{2}$$

where u and $\mathbb{P}$ are the fluid velocity and hydrostatic pressure, μ is the fluid dynamic viscosity , and K is the permeability of the porous medium. The last quantity depends on the structure of the solid matrix. We will assume that the medium is isotropic, $K = kI$, where k is an averaged quantity. Since in the Darcy's equations viscous stresses on the fluid are neglected and only the damping force of the porous medium is considered, it is valid for small permeabilities.

In the case of 3D medium, the computational domain is a regular grid on which a random hydraulic conductivity field k is generated. This random hydraulic conductivity field K follows a stationary log-normal probability distribution $Y = ln(k)$, which is defined by a mean m_Y and a covariance function $C_Y(x, y, z) = \sigma_Y exp(-[(\frac{x}{\lambda_x})^2+(\frac{y}{\lambda_y})^2+(\frac{z}{\lambda_z})^2]^{\frac{1}{2}})$ where σ_Y is the variance of the log hydraulic conductivity and λ_x λ_y and λ_z) are the directional correlation length scales in each direction. To generate the random

hydraulic field, a spectral simulation based on the FFT method (Fast Fourier Transform method) is used. For the sake of simplicity, we take the same value λ for λ_x, λ_y and λ_z .

The physical modeling of the heterogeneous media leads to several types of difficulties even for the linear Darcy equation. The treatment of various scales leads to solving sparse linear systems of very big size with bad condition number due to the heterogeneous permeability K.

The σ and the λ parameters play on the stiffness of the linear system to be solved. The range of σ^2 is usually from 2 to 6 and the λ goes from 2 to 10. σ plays on the amplitude of the permeability K, for $\sigma^2 = 4$ the K varies in mean from $10^{-7.28}$ to $10^{7.68}$. λ plays on the length scale for the change of K, smaller is λ greater is the probability that the K vary strongly from cell to cell.

2.2 The Schwarz algorithm

The domain Ω is split in the z direction into overlapping macro-domains Ω_i. the domain is discretized with regular step sizes in each direction. The Schwarz algorithm consists to update the macro-domain boundary conditions until convergence with taking values in the neighbors macro-domains.

Defining A_i the discrete operator of the Darcy equation on the macro-domain Ω_i that takes into account the Dirichlet boundary conditions on the two artificial interfaces $\Gamma_{i,l}$ and $\Gamma_{i,r}$, then the multiplicative Schwarz algorithm writes:

$$\left\{\begin{array}{l} A_{2i+1}p_{2i+1}^{2n+1} = f_{2i+1}, \\ p_{2i+1|\Gamma_{2i+1,l}}^{2n+1} = p_{2i|\Gamma_{2i+1,1}}^{2n} \\ p_{2i+1|\Gamma_{2i+1,r}}^{2n+1} = p_{2i+2|\Gamma_{2i+1,r}}^{2n} \end{array}\right. \qquad \left\{\begin{array}{l} A_{2i}p_{2i}^{2n+2} = f_{2i}, \\ p_{2i|\Gamma_{2i,l}}^{2n+2} = p_{2i-1|\Gamma_{2i,1}}^{2n+1} \\ p_{2i|\Gamma_{2i,r}}^{2n+2} = p_{2i+1|\Gamma_{2i,r}}^{2n+1} \end{array}\right. \tag{3}$$

A parallel solver can be applied to solve the local problem each macro-domain. The convergence of this two level domain decomposition is purely linear . Then the convergence to the solution can be accelerated by the Aitken formula at the artificial interfaces as follows.

Defining $V^l = \left\{p_{2|\Gamma_{2,l}}^{2l}, p_{2|\Gamma_{2,r}}^{2l}, \ldots, p_{2m|\Gamma_{2m,r}}^{2l}\right\}$, the pure linear convergence writes: $\exists \mathbb{P}$ independent of l such that

$$V^{l+3} - V^{l+2} = \mathbb{P}(V^{l+2} - V^{l+1}) \tag{4}$$

The Aitken acceleration of the convergence writes

$$V^{\infty} = (I - \mathbb{P})^{-1}(V^{l+3} - \mathbb{P}V^{l+1}) \tag{5}$$

Once the converged solution is obtained at artificial interfaces, one local solve gives the solution on the macro-domain.

2.3 Explicit building of $\mathbb{P}$ and Adaptive Aitken-Schwarz

Let $\Phi = \{\Phi_{jk}\}_{j\in[0,\ldots,N],k\in[0,\ldots,N]}$ be a set of orthogonal vectors with respect to a discrete Hermitian form $[[.,.]]$. One can consider without loss of generality that these vectors satisfy $[[\Phi_{jk}, \Phi_{jk}]] = 1, \forall j,k \in [0,\ldots,N]$.

Consider the decomposition of the trace of the Schwarz solution $p_{i|\Gamma_{i,\{l,r\}}}$ with respect to this orthogonal set: $p_{i|\Gamma_{i,\{l,r\}}} = \sum_{k=0}^{N}\sum_{j=0}^{N}\alpha_{jk,\{l,r\}}\Phi_{jk}$. with $\alpha_{jk} = [[p_{i|\Gamma_{i,\{l,r\}}}, \Phi_{jk}]]$. Then the nature of the convergence does not change if we consider the error coefficients in the basis $\{\Phi_{jk}\}_{j\in[0,\ldots,N],k\in[0,\ldots,N]}$ instead of the error in the physical space. One can write the error components iterations equation but in the coefficient space.

Let $\hat{V}^l = \left\{[[p_{2|\Gamma_{2,l}}^{2l}, \Phi]], [[p_{2|\Gamma_{2,r}}^{2l}, \Phi]], \ldots, [[p_{2m|\Gamma_{2m,r}}^{2l}, \Phi]]\right\}$ Then, we can write in the coefficient space:

$$\hat{V}^{l+3} - \hat{V}^{l+2} = \hat{\mathbb{P}}(\hat{V}^{l+2} - \hat{V}^{l+1}) \tag{6}$$

This matrix $\hat{\mathbb{P}}$ has the same size as the matrix $\mathbb{P}$. Nevertheless, we have more flexibility to define some consistent approximation of this matrix, since we have access to *a posteriori* estimate based on the module value of the Fourier coefficients.

The explicit building of $\hat{\mathbb{P}}$ consists to compute how the basis functions Φ_{jk} are modified by the Schwarz iterate. Figure 1 describes the steps for constructing the matrix $\hat{\mathbb{P}}$. Step (a) starts from the the basis function Φ_k and gets its value on the interface in the physical space. Then step (b) performs two Schwarz

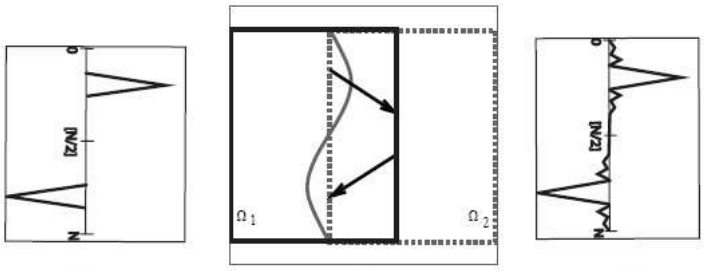


Figure 1: Steps to build the $\hat{\mathbb{P}}$ matrix

iterates with zero local right hand sides and homogeneous boundary conditions on the others artificial interfaces. Step (c) decomposes the trace solution on the interface in the basis Φ. Thus, we obtains the column k of the matrix $\hat{\mathbb{P}}$. The full computation of $\hat{\mathbb{P}}$ can be done in parallel, but it needs as much local subdomain solves as the number of interface points (i.e the size of the matrix $\hat{\mathbb{P}}$). Its adaptive computation is required to save computing. The Fourier mode convergence gives a tool to select the Fourier modes that slow the convergence and have to be accelerated.

3 PARALLEL IMPLEMENTATION

We consider the 3D Darcy equation with Dirichlet boundary condition in X and Y direction and Dirichlet boundary conditions in Z where the domain is split. The domain is a cube $[0, \pi]^3$ with a regular step size discretizing. The difference between two Schwarz iterates satisfies homogeneous Dirichlet B.C. in X and Y. This leads to have $\phi_{jk} = \sin(j\,x)\sin(k\,y), j \in [1, \ldots, N], k \in [1, \ldots, N]$.

The two-level parallelization is performed by creating a 6D cartesian topology under the Message Passing Interface (MPI). The first level consists of splitting the domain, whatever the number of directions, into subdomains called Macro-domains. The second level consists of splitting a Macro-Domain, whatever the number of directions, into subdomains.
The mesh is built setting-up the number of Macro-domains and the number of subdomains. We specify the size of one subdomain.
The following figure presents a domain decomposition for 2 Macro-domains in Z direction, and 2 subdomains in Z, Y and X directions. Processors are enumerated following a global MPI process distribution.

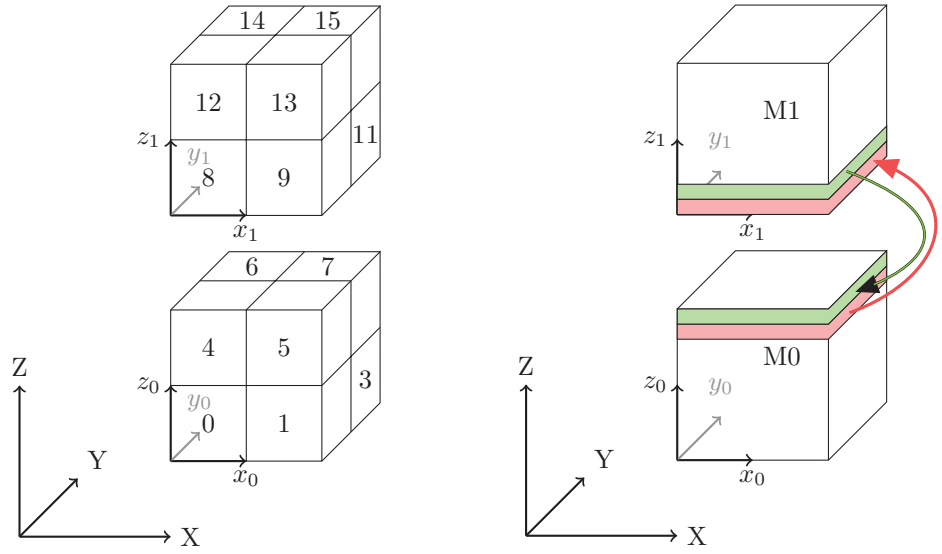


Figure 2: (left) 6D MPI cartesian topology. (right) Exchange of faces between Macro-domains for Z direction splitting

So, the Schwarz method can be set-up choosing how to use the mesh.

3.1 Schwarz exchange

The Schwarz exchanges for a 3D problem involve the sent of a face of a Macro-domain which is distributed on several subdomains. Two issues can be considered.
The first one is based on a totaly distributed program where each subprocess which computes a part of a face of a Macro-domain communicates with the subprocess of the Macro-domain neighborhood which compute the corresponding part of a face. The number of exchange and the time spent for the communications increase with the number of subprocesses on a face of a Macro-domain but the size of the data transfer is smaller than a global sent.
The second possibility is to store a face on such a main process of a Macro-domain and exchange a global interface between Macro-domain. Then, the number of exchange is minimized but the size of the data transfer is larger as the size of a Macro-domain face.

For this study the choice of a global sent between Macro-domain has been done, motivated by the fact that one Macro-domain can be handled by one cluster and communicate with another Macro-domain through a cable network.

We build a Macro-Domain decomposition with overlap 1 and overlap 0 between subdomains. The first implementation consists of splitting the domain into Macro-domains in Z-direction and splitting Macro-domains into subdomains in all directions. A face is collected on the processor 0 of a Macro-domains. Each processor 0 of a Macro-domain can communicate with the processor 0 of a Macro-domain of its neighborhood.

3.2 Solving macro-domains problem

The problem is splitted between Macro-domains with an overlap. Each Macro-problem can be considered as an independent problem. One Macro-domain can be sliced in all the directions without overlap.
Then parallel solver like MUMPS can be used with a distribution of data between Macro-domain's processors. The processor 0 of a Macro-domain deals with the right hand side and the solution. This is the default of the present implementation.
A logical factorization is saved before a solving phase of MUMPS. We used this factorization to solve problem with several right hand side to perform the method presented in the previous section. The first column of the right hand side for the real problem, the other columns for each Fourier modes.

3.3 FFTs

The method for non separable operator needs Fourier Transform or Sinus transform. The parallelization of this phase will be discussed. However the first way is to developp one FFT per Macro-Domain.

4 SCHWARZ ON 3D DARCY FLOW PROBLEM WITH STRONG VARIATION IN THE PERMEABILITY FIELD - FIRST RESULTS

The following study is based on the 3D Darcy flow equation with strong variation in the permeability which is developped in the first section. The Physical domain is a cube of size π. We set Dirichlet B.C. as $-10(e^{-(x-\frac{\pi}{2})^2} - e^{\frac{\pi^2}{4}})(e^{-(x-\frac{\pi}{2})^2} - e^{\frac{\pi^2}{4}})$ at z=0.0 and 0 at $z = \pi$.
We set the parameter for the K generation considering the first section:$\sigma^2 = 4$ and $\lambda = 5$. The domain is sliced in the Z direction both for the first parallelization level and in three dimension for the second, with 2 Macro-domains and 8 subdomains per Macro-domain. For a slab, we set the local number of discretization point in each direction as $nz = 17, ny = 66, nx = 66$. The global grid has 66 points in the X, Y and Z directions.
The following figures show the permeability and the solution on center planes. The third picture shows the linear Schwarz convergence for 2500 alternative steps on the interface and the convergence after 3 alternative steps where $\hat{\mathbb{P}}$ is fully computed after 2 steps.
These results were obtained using 16 processors of a SGI-Altix 350.

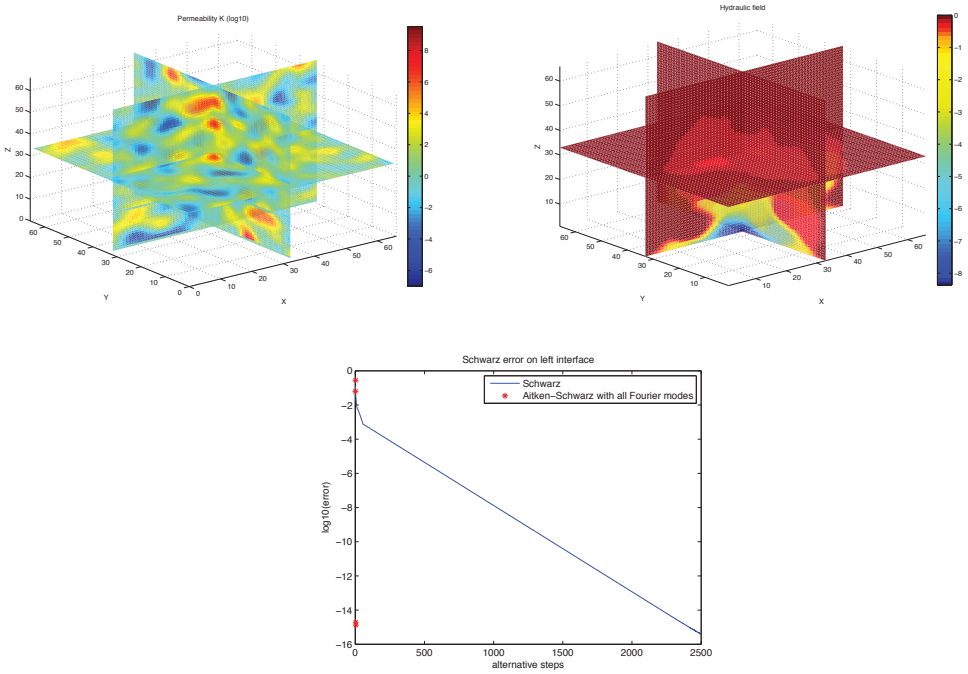


Figure 3: (left corner) Permeability K (log10). (right corner) Hydraulic field. (down) Schwarz error on interface.

5 CONCLUSION

The point of this study is to developp a method which can deal with non separable operator for large problem. The Schwarz method is accelerated by Aitken. The matrix of acceleration is built adaptively with taking the transformation of Fourier basis with the Schwarz avoiding some matrix inverting in the building.
A two-level parallelization is implemented. One level is devoted to the Schwarz method and can be used to send large messages through the network. The lower level is used for the solving phase handeled by a Macro-domain.
A first test case was implemented and run on 2 Macro-domains of 8 subdomains, which represents 16 processors of the SGI-Altix 350 we use. We can observe Schwarz linear convergence on interface. Results will be presented on large parallel machine.

Acknowledgement: This work was funded by the French National Research Angency through projects ANR-07-TLOG-011-03 LIBRAERO and ANR-07-CIS7-004-03 MICAS, and the cluster ISLES/CHP of the region Rhône-Alpes.

References

[1] M. Garbey and D. Tromeur-Dervout, *On some Aitken like acceleration of the Schwarz method.*, Internat. J. Numer. Methods Fluids 40(12), 1493–1513, 2002.
[2] A. Frullone, D. Tromeur-Dervout, *A new formulation of NUDFT applied to Aitken-Schwarz DDM on nonuniform meshes*, Parallel Computational Fluid Dynamics 2005, 493–500, 2006.
[3] Nicolas Barberou, Marc Garbey, Mathias Hess, Mickael Resch, Tuomo Rossi, Jari Toivanen, and Damien Tromeur-Dervout. Efficient meta-computing of elliptic linear and non linear problems. *J. of Parallel and Distributed Computing*, (63(5)):564–577, 2003.
[4] M. Garbey, *Acceleration of the Schwarz Method for Elliptic Problems.*, SIAM J. Sci. Comput. 26(6), 1871–1893, 2005
[5] D. Tromeur-Dervout, *Meshfree Adaptative Aitken-Schwarz Domain Decomposition with application on Darcy flow*, Parallel, Distributed and Grid Computing for Engineering, B.H.V Topping and P. Ivanyi editors, Saxe-Coburg Publications, p.217–231, 2009.

Parallel performance of the Deflated Conjugate Gradient

R. Aubry, G. Houzeaux and M. Vázquez
Corresponding author: romain.aubry@bsc.es

Barcelona Supercomputing Center
29 Jordi Girona
08034 Barcelona, Spain

April 29, 2009

Abstract

1 Introduction

This paper presents the performance of the Deflated Preconditioned Conjugate Gradient (DPCG) solver on a distributed memory supercomputer. When using fractional step techniques, or equivalently pressure Schur complement based techniques to solve the incompressible Navier-Stokes equations, we end up with a symmetric algebraic system for the pressure. In many applications, this pressure solver is the most time consuming part of the overall solution strategy. In addition, the parallel performance of symmetric solvers, like the classical Conjugate Gradient (CG), is very poor as it involves mainly matrix-vector multiplication and scalar products and very few floating point operations apart from these operations. The deflated conjugate gradient has been known now for a decade and hasn't received much attention, despite its extremely nice properties. In the next section we will present the numerical context. In the next section we will introduce the DPCG and present some examples showing its performance compared to the simple CG. Finally, some details of its parallel implementation will be given.

2 The governing equations

When discretizing the Navier-Stokes equations using a numerical method, one obtains a coupled system for the velocity $\mathbf{u}$ and pressure $\mathbf{p}$ to be solved at each time and linearization step:

$$\begin{bmatrix} \mathbf{A}_{uu} & \mathbf{A}_{up} \\ \mathbf{A}_{pu} & \mathbf{A}_{pp} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_u \\ \mathbf{b}_p \end{bmatrix}.$$

Matrix $\mathbf{A}_{uu}$ includes the Galerkin as well as the stabilization terms, like the SUPG-like term and the continuity enforcing term. Matrix $\mathbf{A}_{up}$ includes the Galerkin pressure term. Matrix $\mathbf{A}_{pu}$ includes the velocity divergence operator as well as the part of the pressure stabilization involving the velocity in the momentum residual. Finally, matrix $\mathbf{A}_{pp}$ includes only the pressure stabilization. Note that this matrix is null if

div-stab elements are used. The solution of this system, using a direct solver or an iterative solver with preconditioning, is referred to as monolithic scheme.

The approach followed in this work is algebraic and is extensively studied in [4]. From system (1), we extract the Schur complement system for the pressure, say $\mathbf{Sp} = \mathbf{b}_s$. Then an iterative techniques are applied to solve this system, based on the simple iteration

$$\mathbf{p}^{k+1} = \mathbf{p}^k + \alpha \mathbf{Q}^{-1}(\mathbf{b}_s - \mathbf{S}\mathbf{p}^k),$$

where $\mathbf{Q}$ is the preconditioner, and α is computed such as the residual is minimized at iteration $k+1$. The resulting algorithm, referred to as Orthomin(1), consists of the following Algorithm.

Algorithm 1 Momentum preserving Orthomin(1) iteration

1. Solve momentum eqn $\mathbf{A}_{uu}\mathbf{u}^{k+1} = \mathbf{b}_u - \mathbf{A}_{up}\mathbf{p}^k$.
2. Compute Schur complement residual $\mathbf{r}^k = [\mathbf{b}_p - \mathbf{A}_{pu}\mathbf{u}^k] - \mathbf{A}_{pp}\mathbf{p}^k$.
3. Solve continuity eqn $\mathbf{Q}\mathbf{z} = \mathbf{r}^k$.
4. Solve momentum eqn $\mathbf{A}_{uu}\mathbf{v} = \mathbf{A}_{up}\mathbf{z}$.
5. Compute $\mathbf{x} = \mathbf{A}_{pp}\mathbf{z} - \mathbf{A}_{pu}\mathbf{v}$.
6. Compute $\alpha = < \mathbf{r}^k, \mathbf{x} > / < \mathbf{x}, \mathbf{x} >$.
7. Update velocity and pressure

$$\left\{ \begin{array}{rcl} \mathbf{p}^{k+1} & = & \mathbf{p}^k + \alpha \mathbf{z}, \\ \mathbf{u}^{k+2} & = & \mathbf{u}^{k+1} - \alpha \mathbf{v}. \end{array} \right.$$

3 Deflated Conjugate Gradient

At each iteration of the previous algorithm, a preconditioner for the pressure should be solved. This system is Symmetric Positive Definite (SPD) as it is basically a Laplacian plus stabilization terms. The conjugate gradient [3] is the method of choice for solving SPD systems in an iterative way. It needs very few memory requirements, which is particularly attractive for three dimensional problems, and may be viewed as a direct method, giving the solution in a finite number of steps in exact arithmetic, although it converges must faster in practice. The CG algorithm generates a sequence $x_1, ..., x_i$ such that:

$$x_i \in x_0 + K_i \tag{1}$$

where $K_i = span\{r_0, Ar_0, .., A^{i-1}r_0\}$ is the Krylov subspace of dimension i generated by the initial residual r_0. At each step, the approximation x_i verifies:

$$\|x - x_i\|_A = \min_{u \in x_0 + K} \|x - u\|_A \tag{2}$$

where $\|u\|_A = (Au, u)^{\frac{1}{2}}$. The classical a priori bound for the error in the A-norm is:

$$\|\mathbf{e}^k\|_{\mathbf{A}} \leq 2\Big(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\Big)\|\mathbf{e}^0\|_{\mathbf{A}} \tag{3}$$

where κ is the condition number of matrix $\mathbf{A}$. However, as decribed in [9], the convergence speeds up as soon as the lowest eigenvalues are 'discovered' by the convergence process, giving rise to a condition number based on the active i.e. the non discovered eigenvalues. Therefore, if some knowledge of the eigenmodes associated to the lowest eigenvalues is at hand, removing them from the spectrum of A would give rise to an increased convergence compared to the classical conjugate gradient process. That is what the deflated conjugate gradient tries to achieve. The paper by Nicolaides [7] opens the way to the deflation methods, where the basic idea is to remove some components of the initial residual which may impeed convergence. As opposed to the multigrid techniques [6], no need for defining a smoother, a coarse grid or a prolongation/restriction operator is necessary. A derivation of the deflated conjugate gradient is conducted in [8] which is close to the classical conjugate gradient. This algorithm has been implemented in this paper. For the sake of completeness, the deflated preconditioned conjugate gradient algorithm is given:

- Compute $W^TAWx_0 = W^Tb$
- Compute $r_0 = b - Ax_0$ and $z_0 = M^{-1}r_0$
- Solve $W^TAWd = W^TAz_0$ and set $p_0 = -Wd + z_0$
- Do until convergence:

$$\begin{aligned}
\alpha_j &= (r_j, z_j)/(Ap_j, p_j) && (4)\\
x_{j+1} &= x_j + \alpha_j p_j && (5)\\
r_{j+1} &= r_j - \alpha_j Ap_j && (6)\\
z_{j+1} &= M^{-1}r_{j+1} && (7)\\
\beta_j &= (r_{j+1}, z_{j+1})/(r_j, z_j) && (8)\\
W^TAWd_j &= W^TAz_{j+1} && (9)\\
p_{j+1} &= z_{j+1} + \beta_j p_j - Wd_j && (10)
\end{aligned}$$

Compared to the classical Preconditioned Conjugate Gradient (PCG), the small matrix W^TAW must be inverted. This is performed with a direct skyline solver. An appealing aspect of the DPCG is that a preconditioner may be applied on top of the deflation independantly. It is seen that in multigrid terms, W^T represents a restriction operator and W plays the role of a prolongation operator. Various methods may be used to build these matrices. Various simple domain decomposition methods are discussed in [1]. In a serial implementation, it is advantageous to store matrix W^TA, as it is much sparser than matrix A. However, in a parallel context, this construction is difficult and involves lots of communications. Furthermore, this has to be performed every time that the matrix changes. In our particular implementation, an matrix-vector multiplication has therefore been added to the algorithm. A straightforward parallel implementation involves a global reduction for the restriction operation needed to compute the right hand side of the small system. This may however become rapidly the bottleneck of the whole process.

4 Results

Figure 1 compares the convergences histories of the CG, DCG and DPCG solver with linelet [5] preconditioner on a 2D example. It consists of the first three time steps of a classical benchmark for turbulence models. This simulation solves the thermal and turbulent flow in a tall cavity with anisotropic meshes [2].

We observe that the CG does not even converge. The DCG does much better, but the use of the linelet speeds up the DPCG dramatically, due to the strong mesh anisotropy.

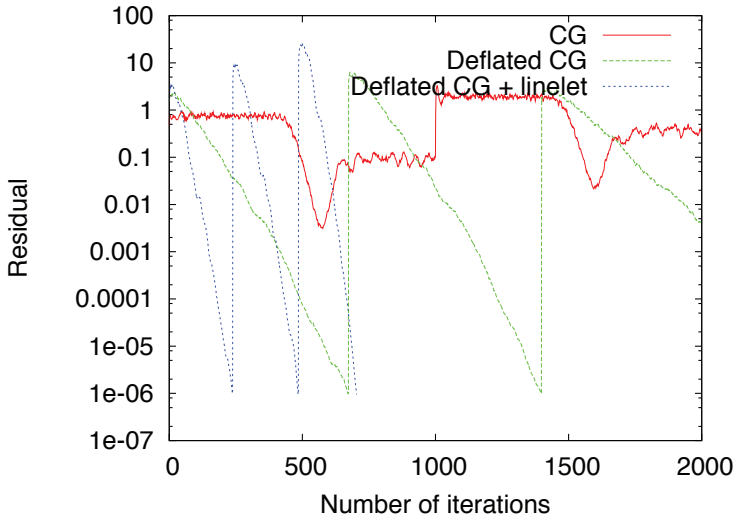


Figure 1: Comparisons of different CG solvers.

5 Conclusion

The Deflated Conjugate Gradient has been presented in this paper for the resolution of the Poisson iterative solver inherent of incompressible algebraically splitted solvers. For the conference, further results in parallel will be shown and details of its implementation will be given. It merely relies on a multilevel parallel method for the implementation to be efficient.

References

[1] R. Aubry, F. Mut, and J. R. C. R. Löhner. Deflated preconditioned conjugate gradient solvers for the pressurepoisson equation. *J. Comp. Phys.*, 227(24):10196–10208, 2008.

[2] P. Betts and H. Bokhari. New experiments on natural convection of air in a tall cavity. In *4th UK National conference on heat transfer, IMechE Conference*, pages 213–217. Mechanical Engineering Publications for the Institution of Mechanical Engineers, 26-27 September 1995.

[3] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.*, 49:409–436, 1952.

[4] G. Houzeaux, R. Aubry, and M. Vázquez. A pressure schur complement solver for incompressible flows: extension of classical fractional step techniques. *In preparation*, 2009.

[5] D. Martin and R. Löhner. An implicit linelet-based solver for incompressible flows. *AIAA-92-0668*, 1992.

[6] S. McCormick. *Multigrid methods*. SIAM, 1987.

[7] R. A. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM J. Numer. Anal.*, 24(2):355–365, 1987.

[8] Y. Saad, J. Yeung, J. Erhel, and F. Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM J. Sci. Comput.*, 21(5):1909–1926, 2000.

[9] A. v. d. Sluis and H. V. d. Vorst. The rate of convergence of the conjugate gradients. *Numerische Mathematik*, 48:543–560, 1986.

A Parallel Free Surface Lattice Boltzmann Method for Large-Scale Applications

Stefan Donath, Christian Feichtinger, Thomas Pohl, Jan Götz, Ulrich Rüde

Chair for System Simulation, University of Erlangen, 91058 Erlangen, Germany
{donath, feichtinger, pohl, goetz, ruede}@informatik.uni-erlangen.de

Abstract: In recent years the lattice Boltzmann method (LBM) has been established as a popular alternative to conventional computational fluid dynamics. With a free surface extension to the method, simulation of bubbly flows and even metal foaming processes is possible. The extension is based on a volume of fluid approach and an explicit tracking of the interface, including a reconstruction of the curvature to model surface tension. In order to accomplish realistic engineering applications, large domain sizes are required, and thus efficient parallelization for several thousand processes is inevitable. Our previous implementation of the parallel free surface algorithm used all-to-all communication schemes resulting in only moderate parallel efficiency when using more than hundred processes. Therefore, the algorithm has been adapted to communicate updates only locally in a restricted neighborhood, which complicates data exchange between processes, in particular when bubbles extend across several subdomains and in case topological changes occur through the coalescence of bubbles. The novel algorithm increases parallel efficiency and enables usage of several thousand processors, rendering large-scale engineering applications like simulation of liquid water in a fuel cell possible. It has been integrated into the waLBerla LBM framework, which features basic tools for communication and data management, designed for massively parallel flow simulations. With this implementation, free surface simulations exhibit parallel efficiency of 90% on up to 4 080 cores.

Keywords: free surface; massively parallel merge algorithm; lattice Boltzmann method.

1. INTRODUCTION

Free surface flow is omnipresent in nature and everyday life. Examples are river flow, oceanic waves, rain, or sparkling water. However, it also plays a crucial role in technological processes. Simulation of many bubbles or of a highly resolved free surface can assist understanding of metal foaming processes (Fig. 1, [8]) or can be used for shape optimization of boats. In order to accomplish such engineering applications, large domain sizes are required, and thus efficient parallelization for several thousand processes is inevitable. Within our seven years of experience in using the lattice Boltzmann method (LBM) with a free surface extension [8, 13], Pohl [9] parallelized the algorithm by means of MPI. Since his method uses all-to-all communication schemes, it achieves only moderate scaling efficiency with more than hundred processes. Therefore, the algorithm has been adapted to avoid all-to-all communication schemes, which complicates handling of topological changes like regions of gas (bubbles) crossing process boundaries or coalescing with each other. The novel algorithm increases parallel efficiency and enables usage of several thousand processors, rendering large-scale engineering applications possible. It has been integrated into the waLBerla framework [2], which is a software suite uniting different LBM flavors with basic tools for communication and data management, designed for massively parallel flow simulations. It is used as platform for sophisticated extensions to the LBM. Besides blood flow and Brownian motion, most prominent examples are multi-component flows, multi-phase or free-surface flows, and flows with moving objects. Most recently, large-scale simulation of fluid-structure interaction (FSI) has been accomplished (Fig. 2, [4]). With the successful integration of large-scale free surface method, simulation in fuel cells (see Sec. 3) is feasible, and possibly the combination with FSI will be achieved in future.

1.1 Related Work

Apart from the LBM, there is a variety of other computational fluid dynamics techniques to study multiphase flow phenomena, such as the volume of fluid method, level sets, and boundary integral methods. However, the advantage of LBM lies in its suitability for flows in complex geometries like porous media, since the mapping to the lattice is

Fig. 1: Simulation of foams with free-surface extension.

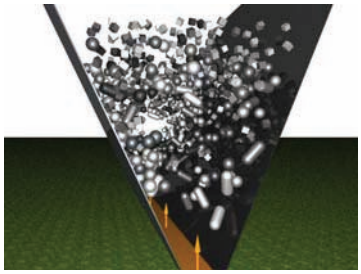


Fig. 2: Parallel simulation of fluidstructure interaction.

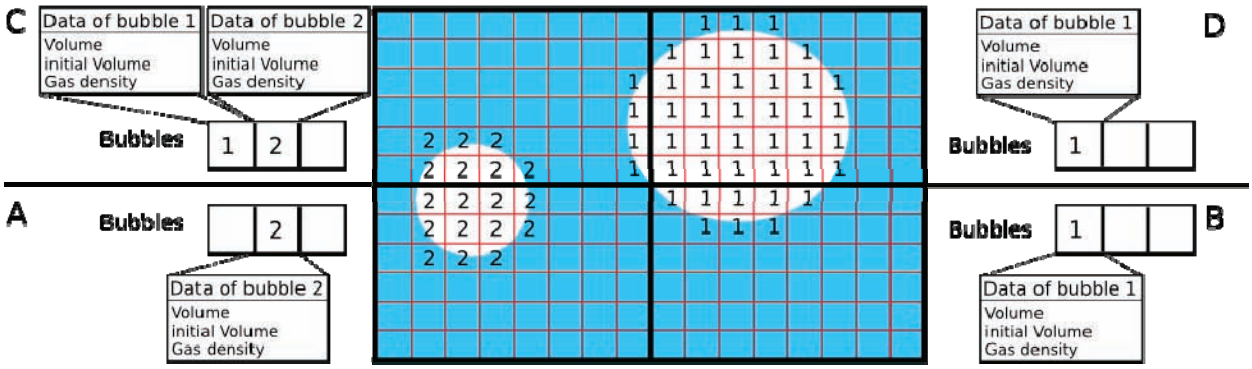


Fig. 3: Data representation of a bubble in free-surface LBM: Gas and interface cells store an ID of the bubble. Additionally, each process stores a list of bubble entries with volume data.

very flexible and involves little computational time. Gunstensen [5] and Tölke [14] presented enhanced variations based on the two-phase approach of Rothmann and Keller [10], which models capillary pressure resulting from the color gradient of a recoloring scheme. Shan and Chen [11] proposed to use an interaction potential representing molecular forces to simulate multiphase and multicomponent fluid flow. There are also methods based on free-energy approach and level sets. Besides the multiphase and multicomponent models, a couple of free-surface models arose in the past decade for simulating moving interfaces between immiscible gas and liquids. The method proposed by Ginzburg and Steiner [3] is the foundation of the variant by Körner et al. [7] which is based on the assumption that the influence of gas phase on liquid phase can be reduced to the force exerted by its pressure and the surface tension. Since only the liquid phase is computed and the interface is a more complex boundary condition, this method saves computation time compared to the two-phase methods mentioned before. Pohl [9] implemented Körner's method for three dimensions and altered the algorithm to enable efficient MPI parallelization.

1.2 Outline of Free Surface Lattice Boltzmann Method

The LBM is a cell-based, local update scheme that performs collision operations on so-called particle distribution functions (PDFs) which represent the fraction of the total mass in each lattice site moving in a discrete direction. Its theory [12] has been proven to resemble a solution of Navier-Stokes equations solving time-dependent, quasi-incompressible flows in continuum mechanics.

The free surface extension introduces different cell types for gas, liquid and the interface in between. While gas cells are omitted, in liquid cells standard LBM is performed. A fill value specifies the quantitiy of liquid in a cell. At the interface, missing PDFs from the gas phase are reconstructed such that velocities of gas and liquid equal, and forces by the liquid, gas pressure, and the surface tension are balanced. Complex calculations compute the pressure of the bubbles from their volume and curvature. This implies, that additionally to cell-based data, each process holds volume information on the bubbles (see Fig. 3).

For details on the LBM in general see [12]. Details on the free surface extension can be found in [7, 9].

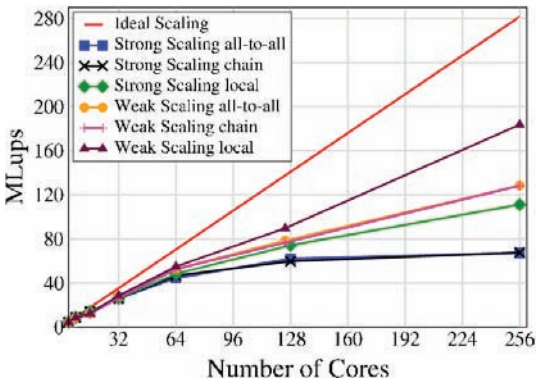


Fig. 4: *Weak and strong scaling on a Woodcrest platform [15] simulating a single small rising bubble in the center of the domain.*

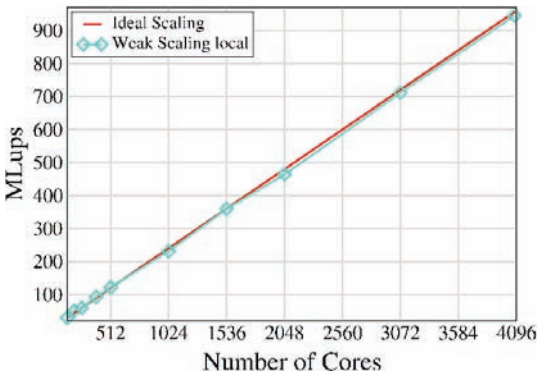


Fig. 5: *Weak scaling on an Itanium 2-based SGI Altix [6].*

2. LOCALIZED BUBBLE MERGE ALGORITHM

In previous implementations, handling of bubble data was not localized. Every process stored volume data of every bubble, irrespective where the bubble was located, and volume changes were communicated in each time step among all processes. Pohl [9] implemented two techniques to achieve this information exchange: Using all-to-all communication primitives of MPI library turned out to be less efficient than arranging all processes in a chain, where the two ends start to send packets upwards and downwards, respectively, and every participant in between merges the own information to the message which is passed on. The latter communication scheme results in *N-1* iteration steps for *N* processes, but involves only direct neighbor communication, minimizing messages over slow connections if processes are properly placed in the network.

In order to enable massively parallel free-surface simulations, the novel algorithm avoids global communication by storing bubble data only on processes that require it. As a consequence, bubbles crossing the boundary of a process have to be sent to the neighbor, and leaving of a bubble has to be recognized to allow deletion of volume data. Moreover, bubble coalescence has to be handled locally among the involved processes of the bubbles, which may become complicated if several bubble volumes have to be merged, possibly with data unknown to a process. The key of the concept is to store a handle for each process a bubble resides on, and communicate this data among all processes of this bubble. Additionally, information on coalescence with other bubbles, including the name of the process that is responsible for the fusion, is stored in the bubble data. With this information distributed, the requirements can be fulfilled: For proper detection whether a bubble crosses the boundary, the cell-based bubble ID (see Fig. 3) is used. If an unknown ID appears in the halo, data for a bubble has to be received. Likewise, if an ID touches the border to a neighbor that is not contained in the process list of this bubble, it has to be sent. At the same time, all other processes knowing this bubble have to be informed on the change. Handling merges is more complicated. The novel algorithm works with variable number of iterations, depending on the complexity of situation. Higher bubble IDs are merged to the lower, and a process responsible to perform a certain merge pair has to wait until all higher merges are done. If a merge is performed, all processes that knew one of the involved bubbles will be informed on the change. Conflicts can arise if bubble merge information is invalidated by another merge. In this case, the algorithm ensures restoration of consistency by rippling messages through the corresponding processes. Since MPI library requires an explicit receive issued in order to establish exchange of information, processes only passively involved in the merge action (i.e. know bubbles but not perform the merge themselves) have to communicate in each iteration until the expected data arrives. More details on this algorithm, including a description of its implementation in waLBerla, can be found in [1].

Consequently, the novel algorithm possibly means more communication in case of merges. However, this communication occurs on a more local vicinity, i.e. the neighborhood of processes harboring the bubbles involved. Since merges of bubbles occur rarely (a bubble usually covers the distance of one cell in approx. 1 000 time steps),

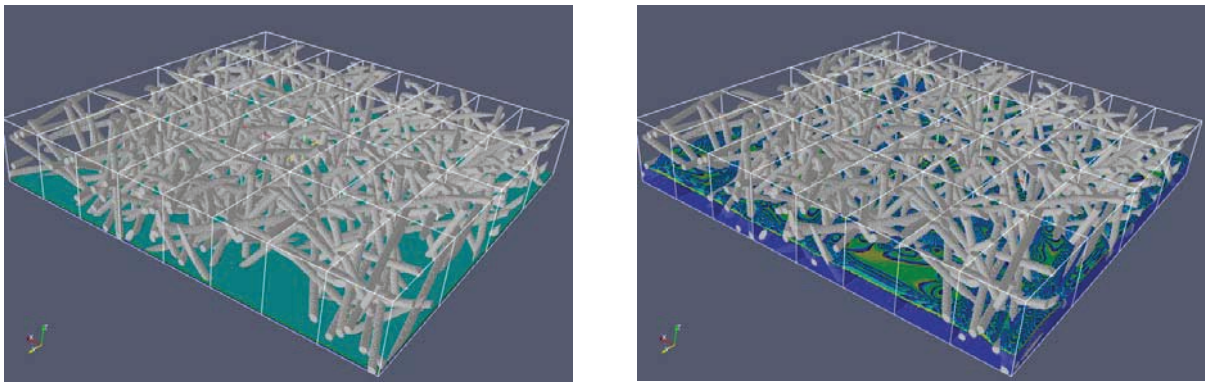


Fig. 6: *Small extract of simulation in fiber geometry similar to gas diffusion layer of a fuel cell. Colors on liquid surface depict fill values.*

the more complicated merge algorithm is paid off by the performance gain due to the saved all-to-all messages. Thus, parallel efficiency benefits by the locality of volume data exchange compared to the all-to-all communication in each time step. Figure 4 gives an impression of performance improvement, while Fig. 5 proves that this algorithm is suitable for large-scale parallel runs on up to 4 080 cores.

3. LARGE-SCALE FREE-SURFACE APPLICATIONS

A recent project with other universities, research institutes, and industry involves the simulation of liquid water in a polymer-electrolyte fuel cell (PEFC) with a proton exchange membrane. On the cathode side of the membrane, reaction of protons, electrons, and oxygen results in liquid water, which is to be evacuated from the reaction zone to sustain electrical performance. Hence, optimization of the structure and properties of the porous membrane is of particular interest. Since experiments cannot accomplish reliable quantification of water throughput in relation to material parameters due to the micron scales, simulation will assist improvement of process. The gas-diffusion layer (GDL) of a PEFC is characterized by a porous structure consisting of many thin fibers. Water generated in the layer below this structure has to evacuate towards a flow channel above. Fig. 6 shows a flooding scenario in a similar geometry, indicating the suitability of the novel algorithm for large-scale simulations. The simulation shown in Fig. 6 was carried out on a domain of $2.7 \cdot 10^7$ lattice cells. Final evaluations will be extended to domain sizes of $2.5 \cdot 10^9$ lattice cells. Due to nondisclosure agreements, Fig. 6 shows a synthetically generated geometry by random placement of fibers, not resembling any characteristics of a true GDL.

Figure 7 shows that the novel algorithm enables finely resolved simulations of many bubbles: A domain of $7.7 \cdot 10^8$ lattice cells contains 3 000 rising bubbles of different diameters, consuming approx. 400 GB of memory.

ACKNOWLEDGEMENTS

This work is carried out within the framework of DECODE, funded by the European Commision (CORDIS project 213295). Parts of this work are funded by the Bundesministerium für Bildung und Forschung under SKALB (grant 01IH08003A) as well as by the „Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen in Bayern" (KONWIHR) via waLBerlaMC. Many thanks to the cxHPC group at the Regional Computing Center Erlangen (RRZE) for their valued support.

REFERENCES

1. Donath, S., Feichtinger, C., Götz, J., Deserno, F., Iglberger, K., and Rüde, U., *waLBerla: On implementation details of a localized parallel algorithm for bubble coalescence*, Tech. Report 09-3, Chair for System Simulation, Univ. of Erlangen, March 2009.
2. Feichtinger, C., Götz, J., Donath, S., Iglberger, K., and Rüde, U., *Concepts of waLBerla Prototype 0.1*, Tech. Report 07-10, Chair for System Simulation, Univ. of Erlangen, 2007.
3. Ginzburg, I., and Steiner, K., Lattice Boltzmann model for free-surface flow and its application to filling process in casting, *J. Comp. Phys. 185* (2003), 61-99.

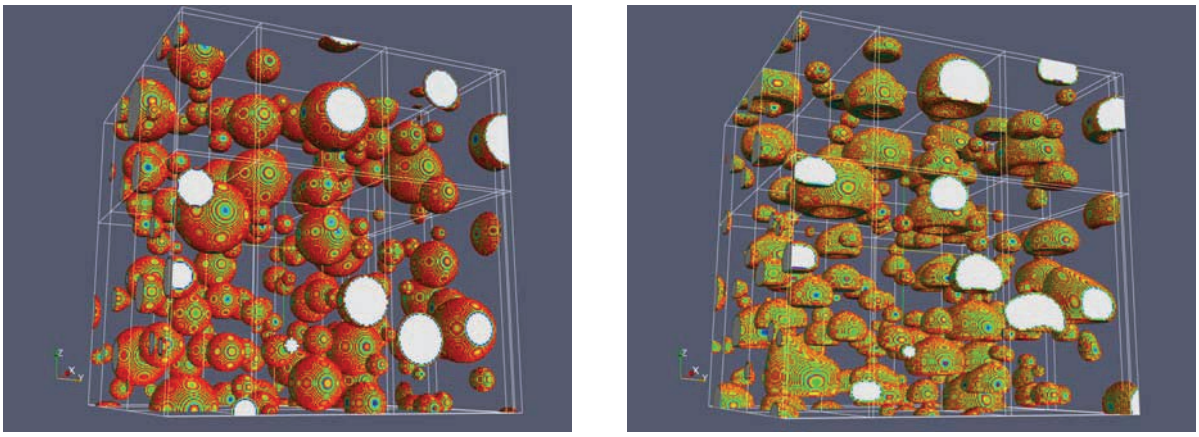


Fig. 7: *Small extract from simulation of 3 000 bubbles at time steps 1 (left) and 15 000 (right) showing only gas cells (white) and interface cells (colored by fill value).*

4. Götz, J., Feichtinger, C., Iglberger, K., Donath, S., and Rüde, U., Large scale simulation of fluid structure interaction using lattice Boltzmann methods and the 'physics engine', *Proceedings of CTAC 2008 (Geory N. Mercer and A. J. Roberts, eds.), ANZIAM J., vol. 50*, 2008, pp. C166-C188.
5. Gunstensen, A.K., Rothman, D.H., Zaleski, S., and Zanetti, G., Lattice Boltzmann model of immiscible fluids, *Phys. Rev. A 43* (1991), no. 8, 4320-4327.
6. HLRB2, http://www.lrz-muenchen.de/services/compute/hlrb/.
7. Körner, C., Thies, M., Hofmann, T., Thürey, N., and Rüde, U., Lattice Boltzmann model for free surface flow for modeling foaming, *J. Stat. Phys. 121* (2005), no. 1-2, 179-196.
8. Körner, C., Thies, M., and Singer, R.F., Modeling of metal foaming with lattice Boltzmann automata, *Adv. Eng. Mat. 4* (2002), 765-769.
9. Pohl, T., *High Performance Simulation of Free Surface Flows Using the Lattice Boltzmann Method*, Ph.D. thesis, Univ. of Erlangen, 2008.
10. Rothmann, D., and Keller, J.M., Immiscible cellular automaton fluids, *J. Stat. Phys. 52* (1988), 1119-1127.
11. Shan, X., and Chen, H., Lattice Boltzmann model for simulating flows with multiple phases and components, *Phys. Rev. E 47* (1993), no. 3, 1815-1820.
12. Succi, S., *The lattice Boltzmann equation for fluid dynamics and beyond*, Oxford University Press, 2001.
13. Thürey, N., and Rüde, U., *Free surface lattice-Boltzmann fluid simulations with and without level sets*, VMV, IOS Press, 2004, pp. 199-208.
14. Tölke, J., *Gitter-Boltzmann-Verfahren zur Simulation von Zweiphasenströmungen*, Ph.D. thesis, Technical University of Munich, 2001.
15. Woody, http://www.rrze.de/dienste/arbeiten-rechnen/hpc/systeme/woodcrest-cluster.shtml.

Integrated Hurricane and Overland Flow Modelling in Parallel Platform

Muhammad Akbar *, Shahrouz Aliabadi **
Northrop Grumman Center for High Performance Computing of Ship Systems Engineering
Jackson State University, MS e-Center, Box 1400, 1230 Raymond Road, Jackson, MS 39204, U.S.A.
** (Tel: 601-979-1839; e-mail: muhammad.k.akbar@jsums.edu)*
*** (Tel: 601-979-1821; e-mail: saliabadi@jsums.edu)*

Abstract: This study focuses on a few aspects of an on-going project at Jackson State University regarding homeland security in the state of Mississippi. The project proposes an integrated tool for multi-scale storm surge and overland flow (flood) forecast due to hurricane, as well as evaluation of the flood damage on coastal infrastructure including transportation systems in the Mississippi coast. Three models are executed in sequence to get all the necessary results. Two out of these three codes are extensively parallel to ensure real time forecast to deal with the emergency evacuation days before the hurricane strikes the coast. The results from the models are fed into Geographical Information Systems (GIS) for visualization, analysis and decision-making.

Keywords: multi-scale hurricane simulation, meteorological data, overland flow, parallel computation.

1. INTRODUCTION

In this study, we present an integrated modelling scheme of a hurricane from its approach to landfall and associated water surge and flooding in the coastal regions. Using the most updated meteorological data days before a hurricane strikes, the ground wind speed, pressure, rain, etc can be predicted using the open source parallel code Weather Research and Forecasting (WRF) [1]. We obtain wind speed and pressure data from WRF, which are used as input to another open source parallel code ADvanced CIRCulation (ADCIRC). ADCIRC is a model for oceanic, coastal and estuarine waters [3]. We use ADCIRC results to model the coastal area flooding phenomena using our finite element method based CaMEL Overland flow solver [3]. The water surge values simulated from ADCIRC along the shoreline is used as the Dirichlet boundary condition input to CaMEL Overland. The rain data predicted from WRF is used as the source term in this solver. The graphical presentation in Fig. 1 shows the integration of the whole simulation process

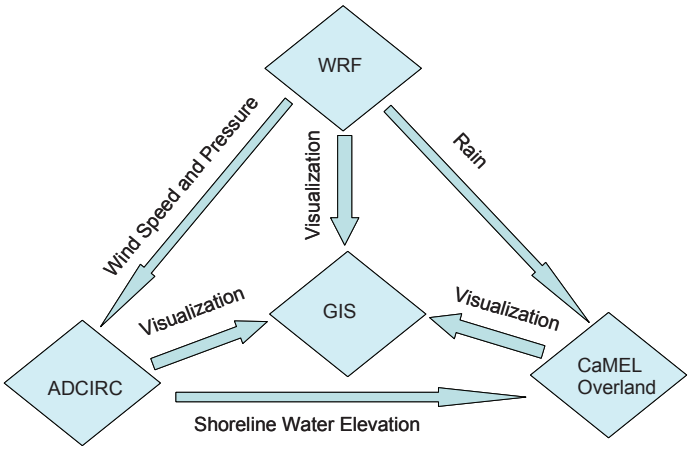


Fig. 1: *Graphical representation of our integrated modelling scheme.*

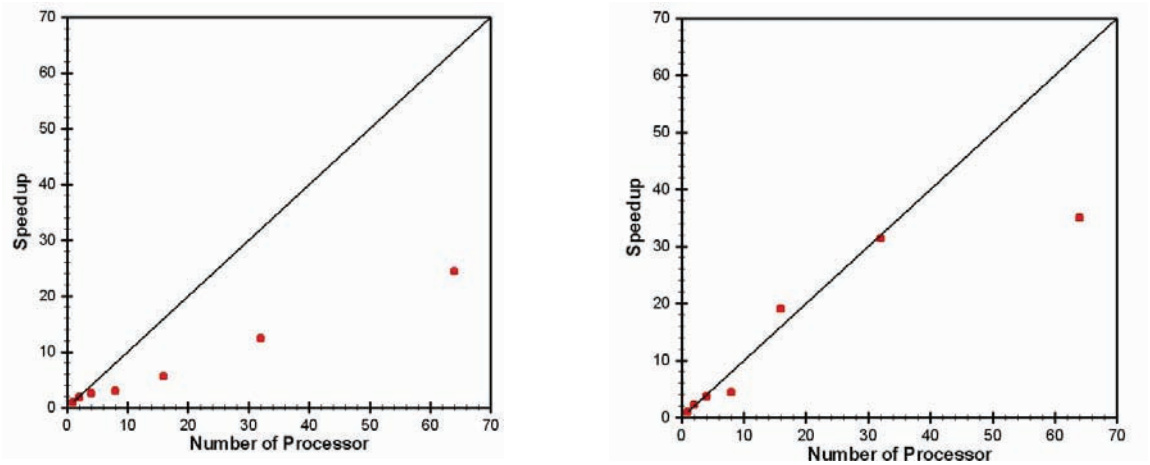
In an actual hurricane case WRF, ADCIRC, and CaMEL Overland codes must be executed in sequence two to three days before its landfall, most likely every 6 to 12 hrs. Repeated simulations of the codes are needed because the more recent meteorological data we use the better accuracy we obtain from WRF. The accuracy of WRF results propagate into ADCIRC and CaMEL Overland simulations through the wind and rain input. Therefore, parallel implementation of the codes is absolutely necessary to ensure real time hurricane and flood forecast.

As a case study in the present research, we have chosen hurricane Katrina (2005) and its flooding impact on the Mississippi coastal region.

2. MODEL IMPLEMENTATION

WRF is a parallel model, which is designed to serve both operational forecasting and atmospheric research needs. It is suitable for a broad spectrum of applications across scales ranging from meters to thousands of kilometers. It allows researchers the ability to conduct simulations reflecting either real data or idealized configurations. We have studied the parallel implementation of WRF extensively in our parallel cluster, which has Intel Xeon processors and total of 80 cores. The speed up of WRF in our cluster is displayed in Fig. 2 (a). WRF uses structured mesh with the option of multiple nested domains. We used a single domain with 300 grid points in east-west, and 220 grid points in south-north. Each segment was 8 km.

Using the WRF wind speed and pressure data as input, ocean water surge is simulated using two-dimensional depth integrated (2DDI) model of ADCIRC. ADCIRC is a highly developed computer program for solving the equations of motion for a moving fluid on a rotating earth. These equations have been formulated using the traditional hydrostatic pressure and Boussinesq approximations and have been discretized in space using the finite element method and in time using the finite difference method. The water elevation is obtained from the solution of the depth-integrated continuity equation in Generalized Wave-Continuity Equation (GWCE) form. The speed up of ADCIRC in our parallel cluster is displayed in Fig. 2 (b). The ADCIRC grid used in our simulation is the same as Mukai et al. [4], which consists of 254,565 nodes and 492,179 elements. ADCIRC Tidal Database [3], Version ec2001_v2d, is used to extract tide data during the Katrina period. Zero-flux boundary conditions are used on the land boundary, and tidal conditions are used in the ocean boundary.



(a) (b)

Fig. 2: *Code speedup with respect to the number of processors in our cluster. (a) WRF, (b) ADCIRC*

After the ADCIRC simulation, we model the coastal area water surge phenomena using our CaMEL Overland flow code. We have solved diffusive wave or Richard's equation, as shown in (1), by the Galerkin finite element method [3]. The time dependent water surge values simulated from ADCIRC along the shoreline is used as the Dirichlet boundary input in the model. The rain data predicted from WRF is used as the source term in the model. Since the execution of CaMEL Overland code is very fast, we have not made any attempt to make it parallel yet. We, however, will make it parallel soon.

$$\frac{\partial h}{\partial t} - \nabla \cdot \left(K \nabla h \right) = \dot{q} \tag{1}$$

where, h, K, $\dot{q}$ are water elevation, diffusion coefficient, and source terms, respectively.

3. RESULTS AND DISCUSSION

WRF simulation results heavily depend on the meteorological data. Hurricane may take unexpected turns, which only the latest meteorological data may reflect. Hurricane landfall location has a huge impact on ocean water surge. Experience suggests that water rises rapidly if the hurricane hits Louisiana coast, most likely due to the converging funnel effect of complicated land structure. On the contrary, hurricane hitting the Alabama coast is most likely to cause much lesser water surge. The computer modelling done by other researchers suggests that the funnel effect in Louisiana area may increase the surge by 20 to 40 percent [5]. This funnel-effect fact is particularly very much applicable for Katrina (Aug 23-31, 2005) type hurricanes with twisted track paths.

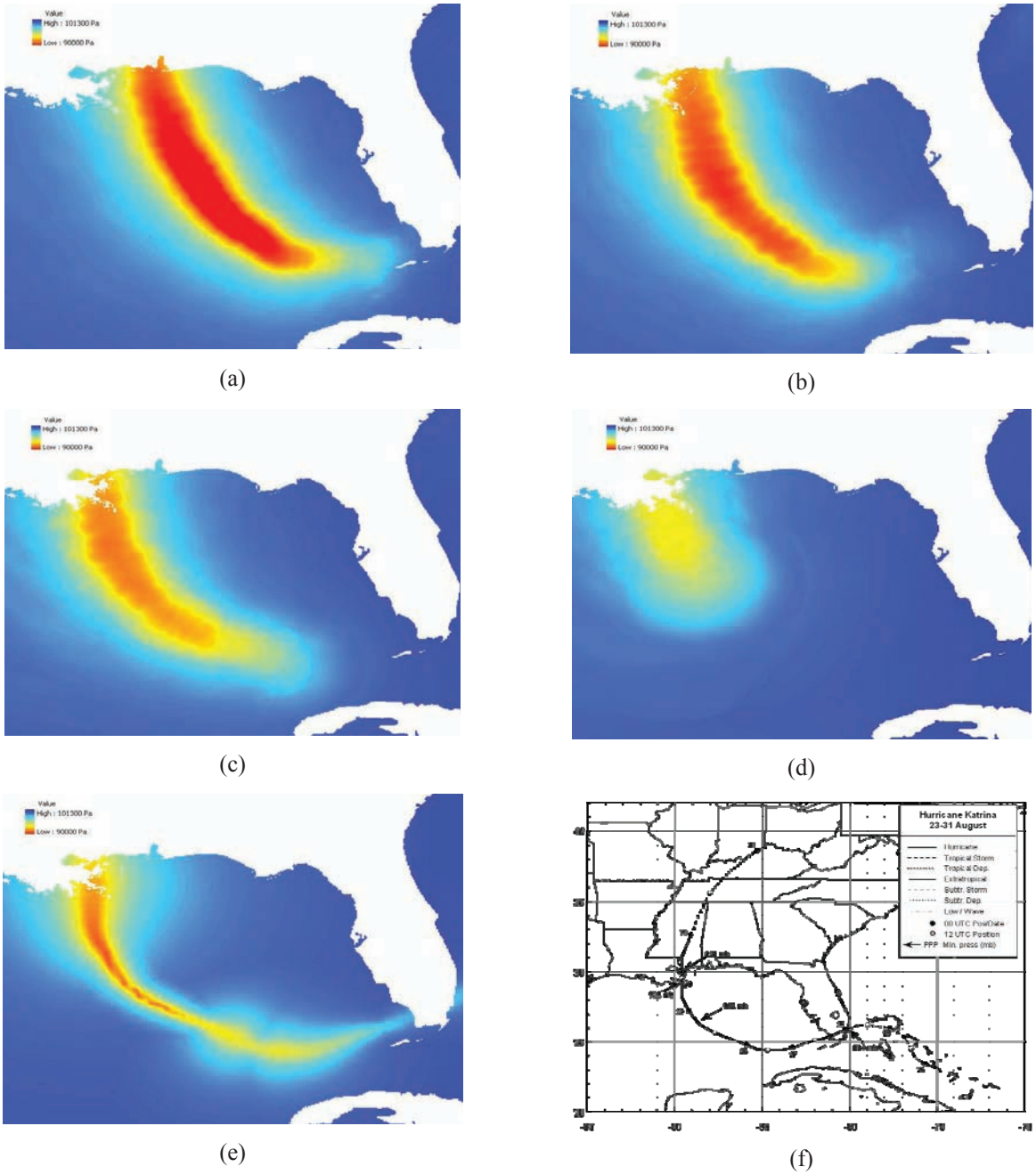


Fig. 3: *Katrina wind pressure plots (a) WRF simulation starting from Aug 26, 00 A.M., (b) WRF simulation starting from Aug 27, 00 A.M., (c) WRF simulation starting from Aug 27, 12 P.M., (d) WRF simulation starting from Aug 29, 00 A.M., (e) Actual, using Planetary Boundary Layer (PBL) (Aug 23 to Aug 31), (f) Published track path.*

Figure 3 shows the comparison of Katrina simulation and actual track path. Figure 3(a), (b), (c), and (d) show the WRF simulated track paths starting from Aug 26 - 00 A.M., Aug 27 - 00 A.M., Aug 27 – 12 P.M., and Aug 29 - 00 A.M., respectively. Figure 3 (e) shows the actual track path obtained by using the Planetary Boundary Layer (PBL). Note that the PBL method interpolates the wind information from the published meteorological data for already past events. The published track path of Katrina is shown in Fig. 3 (f). From the comparison with Fig. 3 (e) and (f), Fig. 3(a) and (b) show that these WRF simulation were started too early. These landfall locations are somewhat east of the actual one. Figure 3(c) appears to have the best result. Although Fig. 3(d) had the latest meteorological data, the hurricane was already too close to the land and it appears to subside. WRF seems to work best with latest meteorological data, while the hurricane is still at least 24 hr far away from the land. This fact underlines the importance of parallel simulation of WRF for quick delivery to facilitate r safe and quick evacuation during an actual hurricane.

Katrina ocean water elevation plots from ADCIRC with different wind speed and pressure input from WRF or Planetary Boundary Layer (PBL) are displayed in Fig. 4. Figure 4 (a)-(c) use WRF wind input with different starting date and time, while Fig. 4(d) uses actual Katrina wind data provided by PBL. From the comparison of Fig. 4(a) – (c) with Fig. 4(d), it is evident that the starting date of WRF simulation has huge impact on the results. It is because of the fact that the latest meteorological data in WRF generates more accurate wind speed, pressure, and landfall location of hurricane. The impact subsequently is carried to ADCIRC and CaMEL Overland codes. In addition to that, ADCIRC simulation has to be done for several days around the hurricane period, typically for 5-7 days, to get reasonably good results. Longer simulation period captures both short and long ocean waves. All the facts mentioned above emphasize the importance of parallel implementation of the codes for real time hurricane forecast to help effective evacuation.

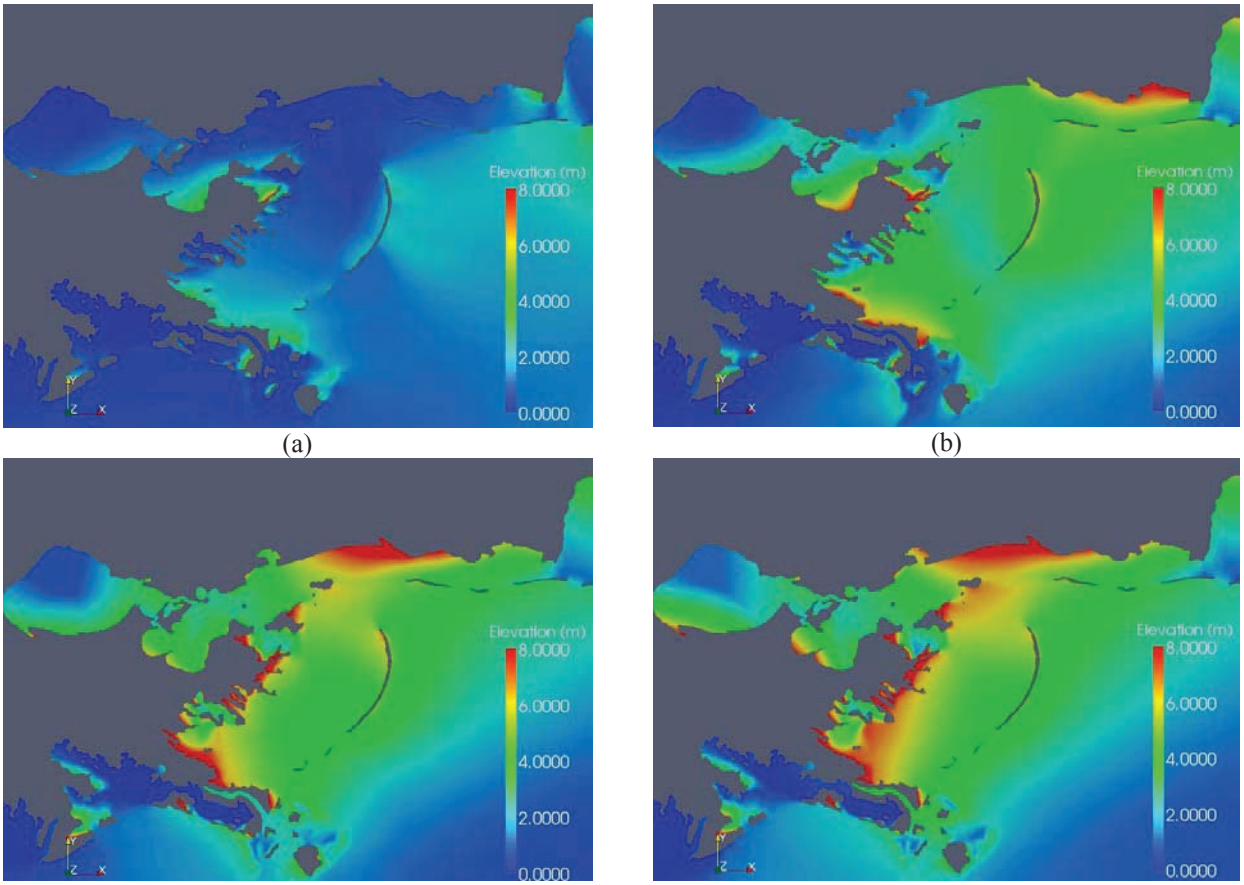
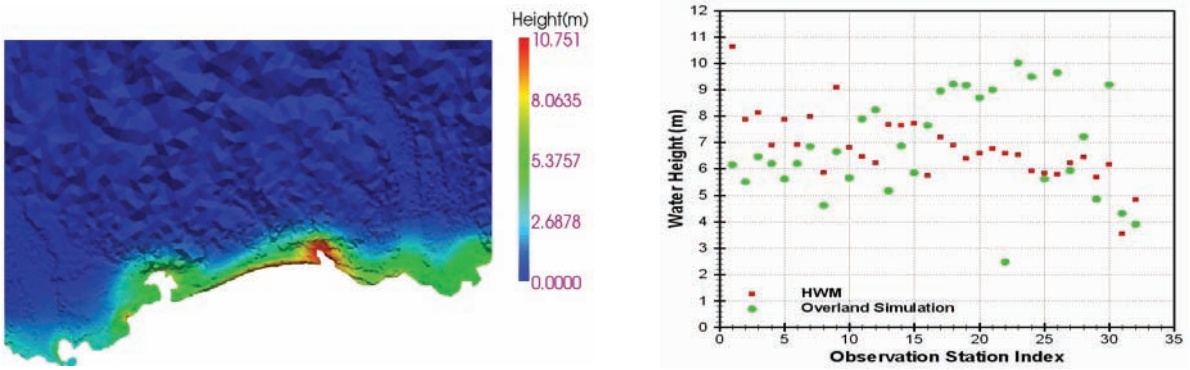


Fig. 4: *Comparison of ADCIRC simulation for different (WRF vs. PBL) wind speed and pressure data as input. (a) WRF - starting date Aug 26, 2005, 00 hr, (b) WRF - starting date Aug 27, 2005, 00 hr, (c) WRF - starting date Aug 27, 2005, 12 hr (d) Actual wind data from PBL (Aug 24 to Aug 31).*

Katrina High Water Mark (HWM) simulated from CaMEL Overland code is displayed in Fig. 5 (a). This result can be used to predict whether any structure in the domain will be flooded, damaged, or unaffected because of the water surge. This is one of the most important information that will help setting up the evacuation plan. This plot is generated using PBL wind speed and pressure data without rain source terms. In real forecast scenario we will not have the actual PBL data available until the hurricane is over. We, therefore, are in the process of using WRF (forecast) data as input in the CaMEL Overland code. Figure 5(b) shows the comparison of simulated Katrina HWM with the observed ones for 32 stations.



(a) (b)

Fig. 5: *CaMEL Overland model Katrina results. (a) HWM in the coastal region, (b) Comparison of observed and simulated HWM.*

4. CONCLUSIONS

We have presented an integrated modelling scheme of a hurricane from its approach to landfall and associated water surge and flooding in the coastal regions using WRF, ADCIRC, and CaMEL Overland codes. We have demonstrated that repeated simulations of the codes are needed because the more recent meteorological data we use, in general, the better accuracy we obtain from WRF. The accuracy of WRF results propagate into ADCIRC and CaMEL Overland simulations through the wind and rain input. We have emphasized the importance of parallel implementation of the codes to ensure real time hurricane and flood forecast for safe evacuation.

ACKNOWLEDGMENTS

This work is sponsored by Department of Homeland Security (DHS) through SERRI project. Authors would like to thank DHS for their support.

REFERENCES

1. Michalakes, J., J. Dudhia, D. Gill, J. Klemp and W. Skamarock: Design of a next-generation regional weather research and forecast model : Towards Teracomputing, *World Scientific*, River Edge, New Jersey, 1998, pp. 117-124.
2. Westerink, J. J., Luettich, R. A., Blain, C. A., & Scheffner, N. W., ADCIRC: An advanced three-dimensional circulation model for shelves, coasts and estuaries. Report 2: Users' Manual for ADCIRC-2DDI. *Technical Report DRP-94*, U.S. Army Corps of Engineers.
3. Akbar, M.K., Aliabadi, S., Wan, T., and Patel, R. "Overland Flow Modeling of Mississippi Coastal Region Using Finite Element Method", Accepted, *19th AIAA Computational Fluid Dynamics Conference*, June 22-25, 2009, San Antonio, TX.
4. Mukai, A. Y., Westerink, J. J., Luettic, Jr., R. A., & Mark, D., Eastcoast 2001, A tidal constituent database for western north Atlantic, Gulf of Mexico, and Caribbean Sea. *U.S. Army Corps of Engineers, ERDC/CHL TR-02-24*.
5. Day, J., Ford, M, Kemp, P., Lopez, J. Mister Go Must Go - A Guide for the Army Corps Congressionally-Directed Closure of the Mississippi River Gulf Outlet. December 4, 2006. (http://www.edf.org/documents/5665_Report%20-%20Mister%20Must%20Go.pdf)

A framework for parallel flow computation with multi-box layout

Kenji Ono*, Takashi Michikawa, Tsuyoshi Tamaki***, Osamu Hiramoto******

*RIKEN, 2-1 Hirosawa, Wako, 351-0198, Japan and Division of Human Mechanical Systems and Design, Faculty and Graduate School of Engineering, Hokkaido University
(Tel: +81-48-467-9469; e-mail: keno@riken.jp)
**RCAST, Univ. of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo 153-8904, Japan
(e-mail: michi@den.rcast.u-tokyo.ac.jp)
*** Fujitsu Nagano Systems Engineering, 1415 Midori-cho, Tsuruga, Nagano, 380-0813, Japan
(e-mail: *ttamaki@jp.fujitsu.com*)

**** HIR, 1406-2 Yanagihara, Nagano, Japan
(e-mail: hira@valley.ne.jp)

Abstract: A novel framework of parallel computing was proposed for inner flows. This approach effectively reduces a number of cells to be computed by eliminating non-fluid cells. A flow channel is covered by many subdomains that aligns the background Cartesian grid. Each subdomain is adjusted such that its workload of computation becomes equal as much as possible. Computation results showed us that the proposed approach scaled up to 128 nodes.

Keywords: Load balance, Partitioning algorithm, scalability

1. INTRODUCTION

One of the problems associated with using CFD for analyzing complex configurations is grid generation; in this context, the energy efficiency and robustness of orthogonal grid method is investigated. In complex configurations in which the scale ratio between the required grid resolution and total calculation space is large, such as industrial piping or a biological vascular network, many unnecessary points that are not used in the calculation are included leading to a decrease in the efficiency of calculat*ion.* AMR (Adaptive Mesh Refinement)[1, 2] is sometimes used to solve these problems and there is excellent software package is already provided[3]; however, in AMR, the mesh ratio at the interface at the boundaries of segments is large, and this has a significant effect on calculation accuracy. When a high-accuracy scheme, which involves the use of a widely spaced stencil, is used, it should be ensured that the width of the mesh comprising the grid is uniform. Given this background, the authors attempted to improve the efficiency of calculations based on an orthogonal uniformly spaced grid by utilizing a bounding box (denoted by bbox), that includes only the part of the grid that is required for the calculation, as much as possible and minimizing the use of the part that is not used in the calculation. The proposed approach ensures scalability for parallel computation by dividing the calculation space into more than one bbox so that the calculation load of each bbox is almost equal. This discourse describes the basic concept and mounting required for the implementation of parallel computation method that is capable of solving the problems described above and demonstrates the effectiveness of the proposed method.

2. PROBLEM ESTABLISHMENT AND GRID GENERATION

The case of a fluid flowing in a series flow path, as shown in Fig.1, is used to evaluate the effectiveness of the proposed method. However, it should be noted that the applicability of the method is not limited to flow path problems. The grid is created using the following procedure.

1. Specification of calculation space

To obtain the calculation space, a bbox that encloses the target object is defined and the global index is corresponded to each axis direction. The width of the grid in each axis direction is the same.

2. Partition of subdomain

(1) The configuration size of subdomain d is specified as $M_d = m_d \times l_d \times n_d$ for three-dimension.

(2) The number of cells in each subdomain is set such that it is identical to that in other subdomains as far as possible (this remarkably affects the parallel execution performance).

(3) It is ensured that the subdomains do not overlap and that each subdomain is always in contact with other subdomains.

(4) It should be ensured that each subdomain covers either the target geometry or a space separated from the target geometry by a specified distance.

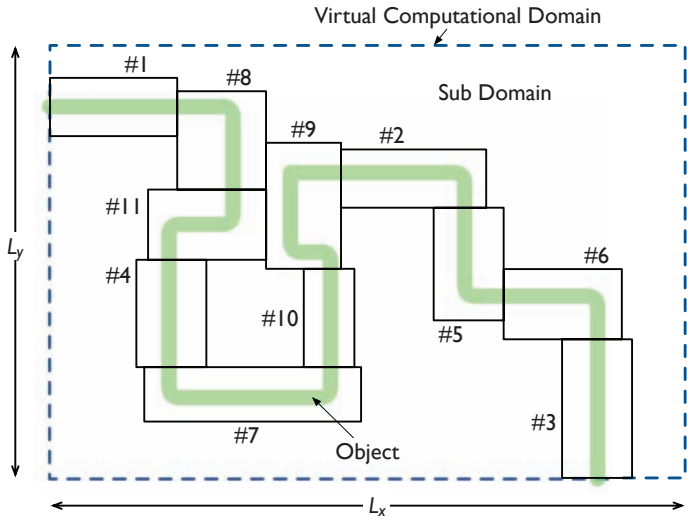


Fig. 1: *Definition of a problem and bounding boxes. In this case, eleven subdomains consist a whole computational domain.*

3. Partition index

(1) An evaluation functionis introduced as valid coverage factor ε_d, which is represented by the ratio between the calculated (fluid) cell number of each subdomain and the configured cell number. The configured cell includes the nonfluid part.

(2) Partitioning is considered as an optimization problem for partitioning the area so that the sum of the valid coverage factors of each subdomain is maximum.

$$\sum_{d=1}^{np} \varepsilon_d \longrightarrow \max \ \textit{and} \ \sum_{d=1}^{np} M_d \longrightarrow \min . \tag{1}$$

The manner in which a subdomain can be defined such that the calculation load becomes uniform and the valid coverage factor attains the maximum value, that is, the manner in which (1) can be optimized, is critical to the calculation.

3. SPACE DIVISION

In order to accomplish the space division of the calculation area, a voxel space for which the configuration is binary approximated, as shown in Fig. 2, is chosen. The voxel approximated by the flow path is defined as a foreground voxel, and the structure the bbox of the subdomain is specified such that the foreground voxel is rendered uniform.

3.1. Problem establishment

If the total calculation space is defined as V and the voxel space of the foreground in V is defined as $f(V)$, which is divided into n subdomains S_i (i = 0, ..., n-1) such that the following conditions are satisfied for the problem of space division.

- S_i should be divided by planes perpendicular to each coordinate axis.

- No two subdomains should overlap.
- S_i should cover $f(V)$ completely.
- The number of foreground voxels in all subdomains should be equal.

In addition to the above conditions, the following nonbinding conditions are also considered.

(1) Max ($f(S_i - \xi)$), that is, the foreground voxel of each subdomain, should be as close to ξ, as possible. ξ will be described later.

(2) The contact area (communications traffic) between the subdomains should be minimized.

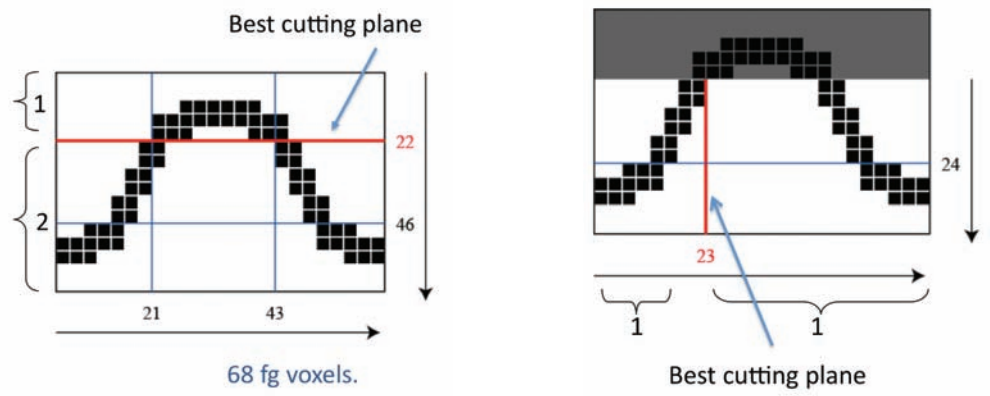


Fig. 2: *Problem setting and algorithm. Voxelization is performed for a fluid path. The described lines show how to find a best cutting plane.*

3.2. Algorithm

Generally, partitioning problems are resolved by excellent algorithms in METIS[4] or Zoltan[5]. However, it is difficult to determine the bbox size and layout simultaneously using those kind of software because they are used to divide a given domain. Therefore, we apply a recursive partition based on kD tree as the partition algorithm. The voxel space A is divided by any plane (xy, yz, or zx) into two partial spaces (B, C). At this point, the number of partitions is set to n and the calculation is performed, as shown below.

$$f(B):f(C) = m:(n-m)\ (0 < m < n) \tag{2}$$

This procedure is repeated until the value of m recursively becomes unity. Fig. 2 left shows examples of a calculation area divided into three parts. In this example, there are 68 foreground voxels. The parts that should be two dimensionally divided by the x plane or y plane are selected using the following equation.

$$Eval(p) = min(sum(p)\%\xi,\ \xi - sum(p)\%\xi) \tag{3}$$

Here, ξ denotes the number of foreground voxels in the target subdomain and is calculated using $\xi = f(V)/n$. sum(p) denotes the sum of foreground voxels from small partition areas. If $\xi = 68/3 \sim 22$ in the example shown in Fig. 2 right, the optimum partition position is at line 22, and the next optimum point is at line 21. In the next step, the remaining area is divided into two parts. When the procedure is repeated for the example shown in Fig. 4, line 23 is

found to be the optimum partition position. This is a simple and robust method, and 99% load balance has been achieved in the exploratory experiment.

4. EVALUATION OF SEGMENTATION AND CALCULATION RESULTS

Figure 3 shows a model used for validation, of which voxel size is 209 x 348 x 114 and the total number of voxels is 8,291,448; further, $f(V)$ = 821,704. In other words, the fluid area to be calculated is approximately 10% of the bbox covering the calculation target. This value decreases if the computational grid width (voxel size) decreases.

In order to evaluate model segmentation equal partitioning was compared with multibox partitioning. Fig. 4 shows the bbox with 128 partitions for the test model. It can be confirmed that the number of voxels to be calculated for multibox partitioning can be decreased to approximately 27% of that to be calculated for equal partitioning at 128 partitions. It is believed that the ratio of nonfluid area decreases and approaches the upper limit, approximately 10% in this case, as the number of partitions increases. Consequently, the number of computational grid points rapidly decreases as the number of nodes increases resulting in a decrease in the calculation time.

The load balance of the fluid cell of each subdomain is shown in Fig. 5. The different numbers of partitions (2 to 60) are considered. This figure shows that 99% or more load balance is achieved. The computer system used is the 64 nodes (128 CPUs) of the RSCC system from RIKEN[6], Japan, which comprises 1024 nodes (2048 CPUs). Fig. 6 shows the ratio of scalability. It is found that the scalability of multibox partitioning is greatly improved compared to that of referential result of equal partitioning, which is approximately 33 times for 128 nodes.

5. CONCLUSION

The authors proposed a parallel computation method that shortens the calculation time by analyzing only the flow path part effectively and by decreasing the ratio of the nonfluid part; this method can be used to compute the flow inside blood vessels or flow paths. Using this method, the effectiveness of parallel computation is improved by utilizing several bboxes that include only the necessary calculation area and by optimizing the load balancing of each bounding box by using a simple and robust algorithm. It was confirmed that by using this method, the calculation time was 20% shorter than that in the case of equal partitioning for up to approximately 128 partitions. This proposed method can be modified to comply with the target, which could be communications traffic, communication pattern, or ratio of the nonfluid cell. In future studies, the case study will be repeated, and higher parallelization will be realized by improving the algorithm for area segmentation.

ACKNOWLEGEMENT

This research was supported by the Research and Development on Next-Generation Integrated Simulation of Living Matter, which is part of the Development and Use of the Next-Generation Supercomputer Project of the Ministry of Education, Culture, Sports, Science and Technology (MEXT).

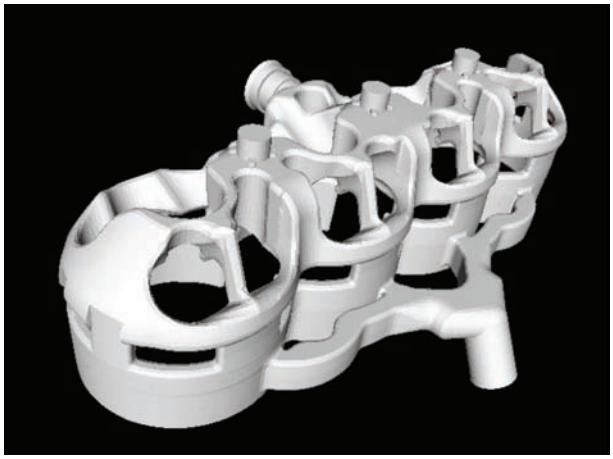


Fig. 3: Test model used for performance evaluation. This shape is a water jacket piping inside an engine block of an automobile.

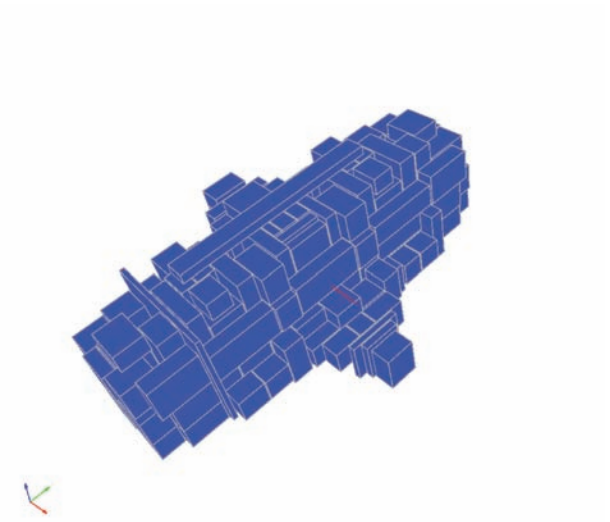


Fig. 4: Comparison of scalability between equal partitioning and multibox partitioning.

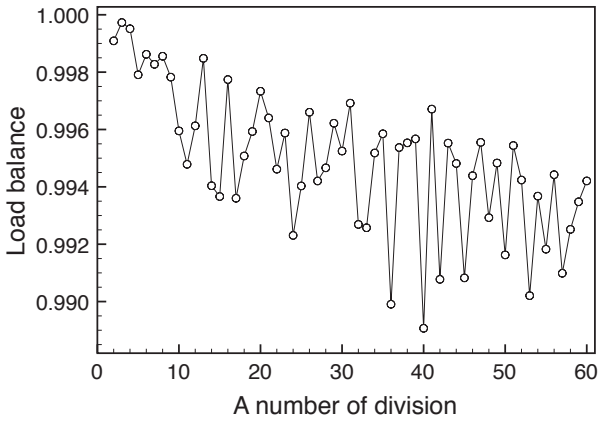


Fig. 5: Load balance of each sub-domain in the case of mulltibox division.

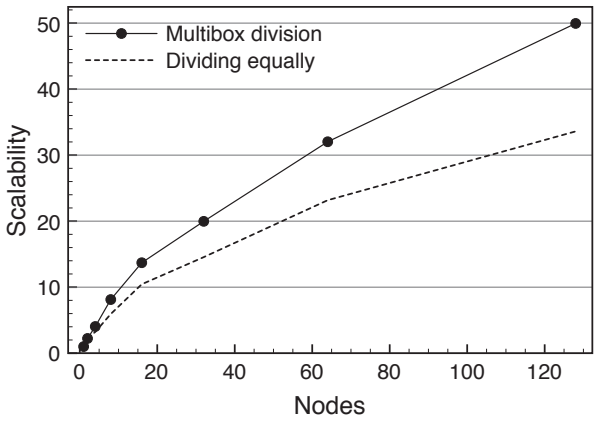


Fig. 6: Comparison of scalability between equal partitioning and multibox partitioning.

REFERENCES

1. Berger, M. and Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations., *J.Comput.Phys,* 53, 484-512.
2. Berger, M. and Colella, P. (1988). Local adaptive mesh refinement for shock hydrodynamics., *J.Comput.Phys,* 82, 64-84.
3. http://seesar.lbl.gov/ANAG/chombo/
4. Karypis, G. and Kumar, V. (1996). Parallel multilevel k-way partitioning scheme for irregular graphs., *Super Computing*.
5. Devine, K., Boman, E., Heaphy, R., Hendrickson, B., and Vaughan, C. (2002). Zoltan Data management services for parallel dynamic applications., *Computing in Science and Engineering*, 4(2), 90-97.
6. http://accc.riken.go.jp/rscc/about_e.html

PARALLEL SOFTWARE DEVELOPMENT

Porting to Cell/B.E. the Alya System, a High Performance Computational Mechanics Code

R. de la Cruz, M. Araya-Polo, M. Vázquez, G. Houzeaux, M. Jowkar and J. M. Cela
Corresponding author: mariano.vazquez@bsc.es

Barcelona Supercomputing Center (BSC)
29 Jordi Girona
08034 Barcelona, Spain

April 29, 2009

1 Introduction

This paper presents a strategy for porting Computational Mechanics (CM) codes to heterogeneus multicore processors, specifically Cell/B.E., whose characteristics make them a very appealing architecture for High Performance CM (HPCM) [3, 4]. The fact is that programming models for these types of processors are still in their infancy. Therefore, based on our own experience with such architectures [1], we introduce in this paper some ideas for re-writting the core parts of a HPCM code in order to take advantatge of the Cell/B.E. processor. The target code is the Alya System [2], which is a BSC in-house multiphysics code, capable of running efficiently in thousands of processors of more traditional shared and distributed memory clusters.

2 The governing equations

HPCM codes are used worldwide to simulate complex problems, where large amounts of total memory and CPU time are required. A typical complex CM problem is governed by one or more diferential equations, coupled, non-linear and with transient behaviour. The governing equations are in turn discretized using a numerical method like Finite Elements, Volumes or Differences, Spectral Methods, etc. which transform the process of finding a solution of the set of coupled differential equations into solving a (potentially very large) algebraic system to obtain an approximate solution, of the desired degree of accuracy. The solution scheme can be either *explicit* or *implicit*, according to the effort needed to invert the system matrix. In the first case, the system matrix is approximated by a diagonal one, which is trivial to invert. In the second case, the matrix is non-diagonal, which can be inverted using, for instance, efficient iterative methods. Then, explicit schemes are fast per-iteration but a large number of time iterations are usually needed to reach the desired solution(s). On the other hand, implicit schemes need less time iterations, but the effort to invert the system matrix could be large. The global turnaround time is what matters: which scheme is the best, strongly depends on the problem. HPCM parallelization effort must then be focused in two sections. The first one is common to both strategies and concern the matrix and right-hand-side computation. The second one, the solver, is exclusive of implicit schemes. In this paper we present results about the former.

Due to both their generality and complexity, the target equations are the Navier-Stokes ones for two kinds of regimes: incompressible and compressible flows. These equations present similar terms, such as convection, diffusion or reaction. They are strongly non-linear and show transient behaviour. In

both regimes, additional quantities can transported by coupling to Navier-Stokes equations, wich will be governed in time by their own convection-diffusion-reaction set of equations. For that reason, the choice of Navier-Stokes equations is a very reasonable one as a starting point to port an established HPCM code to Cell/B.E. In this paper we present preliminary results of the strategy applied to a scalar convection-diffusion-reaction equation with the same kind of terms of the Navier-Stokes'.

Leaving aside some technicalities, an integral form of the equations is the base of well-known numerical methods like FE or FV. Although in this paper we choose FE, most of the discussion can be related to other methods. If these equations are discretized in space using FEM, the system matrix and right-hand-side will be assembled by looping over one of the following choices: elements, faces, edges or nodes. Let us choose the elements as looping units. Before going ahead with the porting strategy, the next section briefly describes the Alya System.

3 The Alya System HPCM code

The Alya System is a HPCM code with two main features. First, it is specifically designed for running with the highest efficiency standards in large scale supercomputing facilities. Second, it is capable of solving different physics, each one with its own modelization characteristics, in a coupled way. Both main features are intimately related, meaning that all complex coupled problems solved by Alya must retain the efficiency. Among the problems it solves are: Convection-Diffusion-Reaction, Incompressible Flows, Compressible Flows, Turbulence, Bi-Phasic Flows and free surface, Excitable Media, Acoustics, Thermal Flow, Quantum Mechanics (TDFT) and Solid Mechanics (Large strain). By specifically designed we mean that Alya is designed from scratch so as to program in a flexible yet clear way every kind of CM model to run on parallel computers. That is to say that Alya was not originally a sequential code which was parallelized afterwards, but designed to be so from scratch. For more information, please visit BSC's CASE Department site [2] and the follow Alya related links.

4 Porting Alya System to Cell/B.E.

In this section, we present the Cell/B.E. architecture, the proposed porting strategy and the preliminary performance results.

Each Cell Broadband Engine (Cell/B.E.) processor (Figure 1.Left) on a QS22 blade, sports a general-purpose 64-bit PowerPC-type Power Processor Element (PPE), which has a traditional cache hierarchy. Eight Synergistic Processing Elements (SPE) are connected to the PPE. The SPEs have a 128-bit wide Single Instruction Multiple Data (SIMD) instruction set, which allows them to process simultaneously four single-precision and two double-precision floating-point operands. Furthermore, the SPEs have software-based scratchpad memories called Local Stores (LS). Detailed hardware specifications are reported in Table 1.

The main difference between generic multicores and the Cell/B.E. architecture is the low latency memory hierarchy. The latter architecture forces the developer to manually manage memory transfers to/from the main memory and LS. Along with this, the following are the characteristics of the Cell/B.E. that we concentrated the most on, during the Alya porting:

1. penalty due to mispredictions of logic branches is very expensive in this architecture, thus reducing control-flow code is mandatory.

2. There are no latency differences in reading and writing from the LS due to its implementation as conventional registers in a processor.

3. To use SIMD instruction, data is required to be aligned and padded properly.

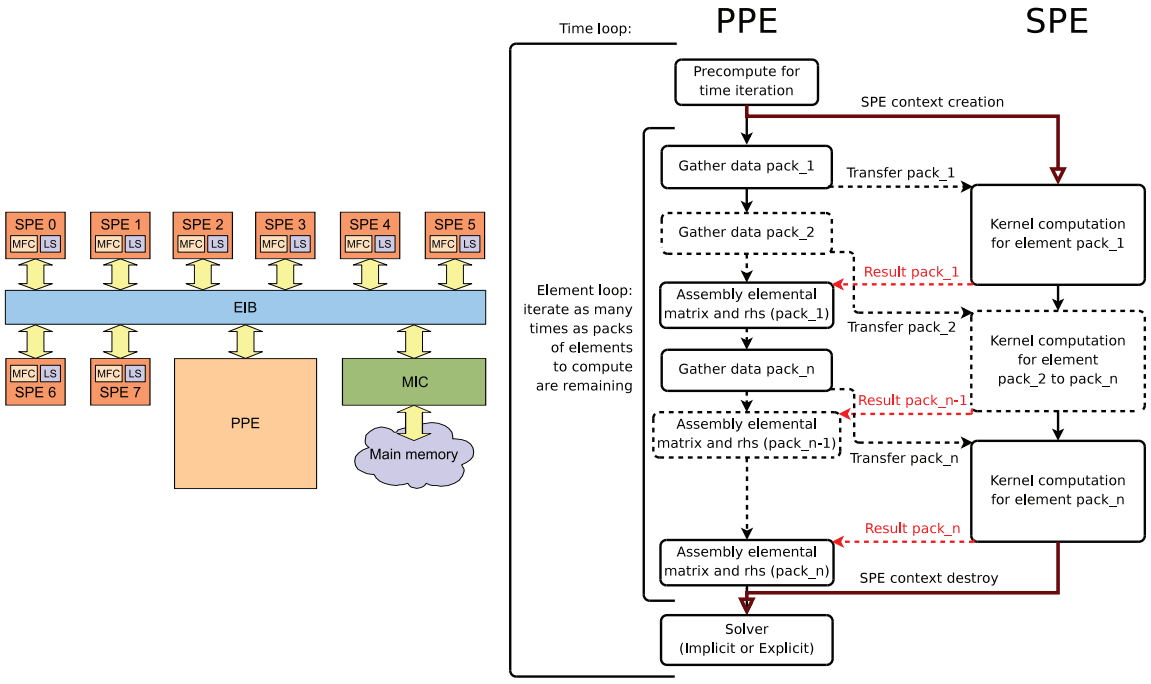


Figure 1: Left: functional block diagram of the Cell/B.E. processor. Right: Alya porting workload schedule scheme.

Blade	JS21 Type 8844	QS22 Type 0793
Processors	PowerPC 970MP	Cell/B.E.
Sockets x cores	2×2	$2 \times$ (1 PPE + 8 SPEs)
Memory per blade (Gbytes)	8	8
Clock Frequency (GHz)	2.3	3.2
SIMD width	128 bit (SP)	128 bit (SP/DP)
Cache memory		
L1 (data + instr)	32k + 64k	32k + 32k
L2	1M per core	512k per PPE
Scratchpad memory	—	256k per SPE

Table 1: Technical specifications of all the systems employed in our experiments.

Given the above characteristics, we will in the next section present an initial implementation strategy for porting an HPCM code like Alya to the Cell/B.E. processor.

4.1 Porting Strategy

The general idea is twofold, to assign the most computational intensive tasks to the SPEs, due to their computing capacity, and to distribute the tasks among the 8 SPEs in a work balanced fashion. Therefore, we have to identify opportunities within Alya's main control flow structure (Algorithm 1) where to apply our porting strategy. The natural parallel structure in Alya is the elemental loop. This loop is composed by five tasks (lines 5 to 9 of the Algorithm 1). Some of these task are suitable to be ported to the SPE, this based on complexity and computational demand. The tasks on lines 6, 7 and 8 are our initial porting targets. On the other hand, the tasks on lines 5 and 9 of the algorithm are strongly tied to data management, and then not suitable for porting to the SPEs. This is due to the complexity of the explicit

memory management.

Algorithm 1 General control flow structure of Alya. Computational cost for each task are included, these costs are evaluated for a certain case, case detailed in the next subsection

```
1: { Time loop }
2: for t = 0 to time_end do
3:    { Elements loop }
4:    for ielem = 0 to nelem do
5:       Gathering and coefficients (3.5%)
6:       Isoparametric and cartesian derivatives (36.5%)
7:       Variables at the gauss points of the element (18%)
8:       Build elemental matrix and rhs (33.5%)
9:       Assembly elemental matrix and rhs (8.5%)
10:   end for
11:   Solver (Implicit/Explicit)
12: end for
```

Data management is a central concern for Alya porting because CM codes that follow FE or FV methods have sparse data structures, coming from graphs of elements and nodes of unstructured space discretizations. The SPEs access to main memory is operated by Direct Memory Access (DMA) lists, which means out-of-cpu and no-coherent control memory read/writes. Sparse data structures are not suitable for SPE mapping, mainly due to two reasons: the DMA list transfers have to access non-contiguous data, causing that each DMA list have to be computed, and the computation of each DMA list involves integer operations (offset and lengths), which are less efficient on the SPEs than SIMD operations.

We propose two different strategies to tackle the data management problem. This strategies are focused to solve the problem for task $6, 7$ and 8, but can be general used for other similar purposes. In the first strategy the PPE gathers data into a sequential buffer, thus avoiding the expensive DMA lists computation, then the SPEs access the sequential buffer using a static DMA list. Second strategy, the PPE precomputes the DMA lists, and subsequently transfers the lists to the SPEs. Then, the SPEs use the precomputed DMA lists to fetch the data. When SPEs computation is finished, the first strategy implies that the SPEs transfer back data to the sequencial buffer and then the PPE scather data to the natural location. The second strategy involve transfers back the results to main memory, and no further action by the PPE.

The strategies comparison criteria are computing, communication and implementation effort, where the former are more important than the latter. The first strategy is expensive in computing (data gather/scather at every time step) but cheap in communication (just two transfer at every time step) and implementation effort. The second strategy is expensive in communication (many transfers) and implementation effort (DMA lists computation for sparse data), but it is cheap in computing terms, the computation of the DMA lists is cheaper than the gather/scather process. With no clear winner both strategies are explored as data management solutions. Figure 1.Right shows an overview of the first porting strategy schedule scheme.

It is important to notice that our implementation is fully SIMDized for the Cell/B.E. architecture to fully exploit the SPEs capacity.

4.2 Performance Evaluation

The following results show the performance of task 6 and 8, which represent 36.5% and 33.5% respectively of the total execution time of the elemental loop (Algorithm 1). The addition of these two task

represent the main part of the execution time, thus performance improvements on these tasks give us a good idea of what will be the performance improvement of the whole code.

The computational cost of each task is evaluated through a test on a 3D canal mesh computed on Alya HPCM code, where 1000 tri-linear hexaedral elements (8 nodes and 8 gauss points per element) were processed. The test case involves the solution of a heat equation, and it was repeatedly executed in order to obtain reliable tasks cost information.

Task line	JS21 Time [ms]	QS22 Time [ms]	Speed-up
6	10.04	0.76	13.21
8	9.22	1.25	7.37

Table 2: Performance results for JS21 and QS22. These results were obtained with *IBM XL Compilers for Multi-core Acceleration*. Notice that only one chip per blade was used during the experiments

Table 2 resumes the performance results. Task 6 is 13.2x and task 8 is 7.37x times faster in QS22 than in the classical multicore architecture JS21. The projected performance improvement for Alya is 10.6x, which matches with our previous experiences [1] on mapping high performance scientific kernels to Cell/B.E. achitecture. The performance gains are basically explained by the implementation of a fully SIMDized code, and the parallel processing achieved by the 8 SPEs of the Cell/B.E. processor.

5 Conclusions and Future Work

This paper presents results on porting a general purpose HPCM code to an heterogeneus multicore like Cell/B.E. We focus first on the matrix and right-hand-side assemblies. Then the porting strategy is discussed and assessed through speed-up figures, which are indeed very promising. The next steps are (1), to explore the best data management strategy solution (2), to extend a similar approach to the solver part of the code and (3), to extend it to different Alya System's modules to solve other Physical problems, like solid mechanics or excitable media.

6 Acknowledgements

This work was supported by "Computación de Altas Prestaciones V" project (TIN2007-60625), Ministerio de Ciencia e Innovación of the Spanish Government.

References

[1] M. Araya-Polo, F. Rubio, M. Hanzich, R. de la Cruz, J. M. Cela, and D. P. Scarpazza. 3D seismic imaging through reverse-time migration on homogeneous and heterogeneous multi-core processors. *Scientific Programming: Special Issue on the Cell Processor*, 16, December 2008.

[2] M. Vazquez and G. Hozeaux. The alya system web page: http://www.bsc.es/csmteam, 2008.

[3] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. The Potential of the Cell Processor for Scientific Computing. In *Proc. ACM Intl. Conf. on Computing Frontiers*, Ischia, Italy, May 2006.

[4] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. Scientific Computing Kernels on the Cell Processor. *International Journal of Parallel Programming*, 35(3), June 2007.

Using XML with Large Parallel Datasets: Is There Any Hope?

Renato Elias*, Vanessa Braganholo, Jerry Clarke***, Marta Mattoso****, Alvaro Coutinho***

**High Performance Computer Center, COPPE/UFRJ, Brazil*
(e-mail: rnelias@nacad.ufrj.br, alvaro@nacad.ufrj.br)
***Department of Computer Science, IM/UFRJ, Brazil*
(e-mail: braganholo@dcc.ufrj.br)
**** US Army Research Laboratory, USA*
(e-mail: clarke@arl.army.mil)
***** Computer Systems and Engineering, COPPE/UFRJ, Brazil*
(e-mail: marta@cos.ufrj.br)

Abstract: This paper discusses data formats to represent large volumetric datasets. We claim that XML-based formats associated with binary formats are appropriate to this scenario because of the self-descriptive nature of XML. Thus, XML files can describe the structure of the binary files, which improves data handling by applications. This also improves interoperability and allows queries to be posed over data.

Keywords: XDMF, Large Volumetric Data Sets, Data Formats.

1. INTRODUCTION

It is well known that High Performance Computing (HPC) applications deal with large volumes of data, which has increased significantly over time. Such large amounts of data are hard to handle, especially in what concerns moving them from one site to another so they can be used by different applications (flow solvers, visualization, etc.). Time spent in moving data through the network could be prohibitive, and then researchers and application developers are trying their best to avoid this as much as possible [11]. To solve this problem, besides developing new access methods and tools, it is essential that data is represented and stored in a way that contributes to this. It has to be, at the same time, compact and self-descriptive, so that no much effort is put to access it. In this paper, we investigate the issue of representing and storing large parallel datasets.

Our main motivation to investigate this issue is data interoperability among several applications for the simulation of oil and gas problems in the Galileo Network, an alliance of five Universities in Brazil developing innovative parallel applications to face new exploration and production challenges in the recently discovered pre-salt areas in ultra deep waters (2,000s meters) southeast of Brazil. These new parallel applications encompass simulation software for computational solid, fluid and structural mechanics that have to share data. Therefore, besides improving data movement between different applications, data formats should enable data interoperability.

There are several ways to represent and store data. HPC applications usually use binary and compressed data provided by libraries such as netCDF, HDF5, ExodusII. They are efficient, but difficult to interoperate among different programs. One of the limitations is its lack of descriptive information on its binary data. Even after finding the required data, accessing it is not simple. This access might be different for each program that interoperates with the binary data. Ross et al. [11] share our point when they say "High-level libraries such as HDF5 and Parallel netCDF have been developed to provide more natural and efficient interfaces when used properly, but adopting these interfaces is a time consuming task, and even with these libraries significant tuning is often necessary". Additionally, parallel simulation applications that use these formats usually require several input files, one for each solution variable, time step and process. For large scale parallel problems, managing the volume and number of files can be prohibitive: several operating systems have difficulties in dealing with many of files in the same

directory. This is thus one more issue that needs to be addressed when choosing a data format.

Text-based formats like XML are self descriptive and flexible but are often taken aside, since they are too verbose and would largely increase the size of the data set, just making things worse. However, the self-description of XML can add consistency, reliability among other desired features to HPC applications. With this in mind, we investigated existing XML-based data formats for HPC applications, and found three main ones: VTK/XML [14], XDMF [1] and XDTM [10].

VTK/XML files can be of two types: structured (for topologically regular datasets such as arrays of pixels or voxels) and unstructured (for topologically irregular sets of points and cells). However, this format does not allow storing heavy data in separate binary files. It encodes all binary data using a Base64 scheme in order to not violate the XML standard. XDTM, on the other hand, has no predefined schema, and thus each application can define their own. The idea is to map XML types into a physical representation. Applications would deal with XML types directly, thus avoiding the difficulties of accessing binary data. The problem of this approach is the need to re-implement applications and no uniformity between schemas. XDMF, on the other hand, classifies data as light and heavy according to the amount of information it represents. Light data is allowed to be stored in the XDMF file's body following the XDMF DTD rules while heavy data is stored in HDF5 format and described in the XDMF file. This approach takes advantage of the XML proposal for a self described data without producing large XML files. XDMF is currently supported by VTK and consequently Paraview and Vislt.

In this work, we present a case study for the XDMF file format in a parallel finite element simulation for a natural convection problem. This problem is a simplified model problem for complex viscous flow and temperature phenomena occurring in salt tectonics [7]. Velocity, pressure and temperature data for several time steps were represented in XDMF and HDF5 for visualization with ParaView. The remainder of this paper is structured as follows. Section 2 presents XDMF in more details. Section 3 presents a case study with XDMF. Finally, Section 4 concludes and discusses future work.

2. XML AND LARGE VOLUMETRIC DATASETS

XDMF stores light data directly in XML format, while heavy data is described in XML but actually stored in HDF5 files. HDF5 alone is not enough, since it is not self-descriptive. The advantage of using XDMF together with HDF5 is that the structure is explicitly represented in the XDMF file, thus making access much easier. In summary, an XDMF file is composed of one or more *Domain*. Domains are composed of Grids, each of which represents a collection of homogeneous elements. Each *Grid* can have temporal data (*Time*), other Grids, descriptions of the data organization (*Topology*), XYZ mesh values (*Geometry*) and mesh values (*Attribute*). Data itself is stored in lower level elements called *DataItem*. There is also a special element called *Information* that can store application specific data.

When HDF5 files are used, DataItem points to positions inside the HDF5 files where data is actually stored. Figure 1 shows an example of XDMF file that points to different portions of an HDF5 file. The example is truncated due to space restrictions, but the following lines repeat the same block pointing to other HDF5 files. Notice that, despite the use of HDF5 the metadata information is explicit in the XDMF file. This makes data access much easier. Moreover, this example demonstrates the concept of spatial collections. Spatial collections can be employed to assemble different geometrical parts of the computational grid. Note that, when working in a domain decomposition context, the domain is spatially partitioned in smaller pieces and individually assigned to MPI processes. In this sense, the XDMF spatial collection is employed to recover the non-partitioned geometrical model. Alternatively, spatial collections could be used to distinguish different geometrical parts of the same model in a serial or parallel run. In this way, a car model could be understood as a collection of spatial parts representing the car doors, tires, wind shield and so on.

Another desirable concept in scientific file formats is the ability to represent transient data. In XDMF this is accomplished by the use of “temporal collections”. In Figure 2 we may observe that a “temporal

collection" is used to list which XDMF file corresponds to each time value. Moreover, in this example, each time step is stored in a different XDMF "spatial collection" file, as previously described and shown in Figure 1, and referenced by the use of *<xi:include href="..."/>* clauses. It gives rise to a file arrangement corresponding to a "temporal collection of spatial collections". Note that, in practice, several possibilities in terms of data storage and arrangements could be made without losing portability and readability across different software and systems. Actually, this is one of the main XML goals and naturally present in XDMF format.

```
<XDMF xmlns:xi="http://www.w3.org/2001/XInclude" Version="2.0">
  <Domain Name="EdgeCFD">
    <Grid Name="foo_00000" GridType="Collection" CollectionType="Spatial">
      <Time TimeType="Single" Value="0.00"/>
      <!-- GRID FOR THE FIRST PARALLEL (SUB)DOMAIN -->
      <Grid Name="foo_000" GridType="Uniform">
        <!-- MESH INCIDENCE -->
        <Topology Type="Tetrahedron" NumberOfElements="   296882 " BaseOffset="1">
          <DataItem Dimensions="1187528 " NumberType="Int" Format="HDF">foo_000_00000.h5:/incid</DataItem>
        </Topology>
        <!-- NODAL COORDINATES -->
        <Geometry Type="XYZ">
          <DataItem Dimensions="1472283" NumberType="Float" Precision="8" Format="HDF">
                    foo_000_00000.h5:/coords</DataItem>
        </Geometry>
        <!-- TRUNCATED LINES -->
      </Grid>
      <!-- GRID FOR THE SECOND PARALLEL SUBDOMAIN -->
      ...
   <!-- TRUNCATED LINES -->
   </Grid>
  </Domain>
</XDMF>
```

Fig. 1: *XDMF for a spatial collection (file foo_00000.xmf).*

```
<XDMF xmlns:xi="http://www.w3.org/2001/XInclude" Version="2.0">
   <Domain Name="EdgeCFD">
      <Grid GridType="Collection" CollectionType="Temporal">
         <Time Type="Single" Value="0.00"/>
         <xi:include href="foo_00000.xmf" xpointer="xpointer(//XDMF/Domain/Grid)" />
         <Time Type="Single" Value="0.01"/>
         <xi:include href="foo_00001.xmf" xpointer="xpointer(//XDMF/Domain/Grid)" />
      <!-- TRUNCATED LINES -->
      </Grid>
   </Domain>
</XDMF>
```

Fig. 2: *XDMF example for a temporal collection (file foo.xmf).*

3. CASE STUDY

The case study presented in this work employs the EdgeCFD software. It is a parallel Fortran90 finite element code for coupled Navier-Stokes and transport problems. EdgeCFD features stabilized and variational multiscale finite element formulations. EdgeCFD is being used for Newtonian and non-Newtonian fluid flows, free-surface flow simulations with interface tracking approaches (volume-of-fluid/level sets), gravity currents and turbulence [see 7 and references therein]. The two equations are solved by a staggered approach. In this software, most of the computational cost comes from the **u**-p coupled solution of the incompressible flow equations while the cheaper part is due to the transport equation. Time integration is a predictor-multicorrector algorithm with adaptive timestepping [11]. Within the flow solution loop, the multi-correction steps correspond to the Inexact-Newton method as described in [3]. As a linear solver, EdgeCFD employs the Generalized Minimal Residual Method (GMRES) since both equation systems, stemming from the incompressible flow and transport, are non-symmetric. Furthermore, a nodal block-diagonal and diagonal preconditioner are used respectively for flow and transport. Most of the computational effort spent in the solution phase is devoted to matrix-vector products. In order to compute such operations more efficiently, we have used an edge-based data structure as detailed by Elias et al [3]. This data structure reduces indirect memory access, memory requirements to hold the coefficients of the stiffness matrices and the number of floating point operations

when compared to other data structures. The computations are performed using the message passing interface library (MPI). The parallel partitions are generated by Metis/ParMetis library, while the information regarding the edges of the computational grid is obtained from the EdgePack library [2]. EdgePack also reorders nodes, edges and elements to improve data locality, exploiting efficiently the memory hierarchy of current processors.

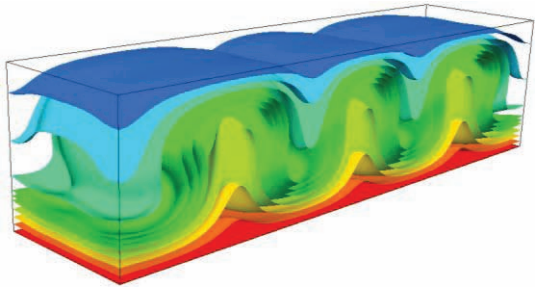


Fig. 3. Iso-temperature contour plots for the three-dimensional Rayleigh-Benard problem.

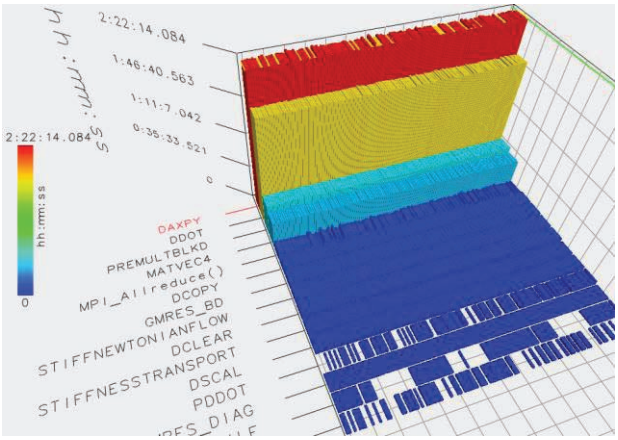


Fig.4. EdgeCFD parallel performance on 128 cores of SGI Altix ICE cluster.

A case study of 3D Rayleigh–Benard convection has been carried out to investigate the XDMF use in a large scale simulation. The benchmark corresponds to a rectangular 3D domain of aspect ratio 4:1:1 aligned with the Cartesian axes and subjected to a temperature gradient [4]. Simulations are made on a 501×125×125 mesh, resulting in 39,140,625 tetrahedral elements using an edge-based SUPG scheme with pressure stabilization. A converged stationary solution, shown in Figure 3, shows the convective rolls obtained at Rayleigh number Ra=30,000 and Prandt number Pr=0.71.This solution was obtained on 128 cores of a SGI Altix ICE 8200 cluster with a fixed time step. Every time step we solve, by the Inexact Newton method, two nonlinear systems of equations, for flow and temperature, respectively with 31M and 7.8M equations.

Code performance has been profiled using TAU on 50 time steps. The three-dimensional execution profile is shown in Figure 4. We may observe the excellent load distribution by the uniform size of the CPU time bars. In this case we used non-blocking point-to-point communication between subdomains. It is worth to emphasize that most of the time is spent in parallel DAXPY BLAS primitives (red bar), which does not involve communication. It indicates that the same problem could be run in much more CPUs without losing scalability.

Following the XDMF concepts, in this case study the heavy information (nodal coordinates, element incidence, pressure, temperature and velocity nodal solution) is hierarchically stored in HDF5 files while all information required to access the HDF5 files (number of elements and nodes) are stored in XDMF files. Moreover, each process stores its own solution temporal collection as described in Section 2, while rank 0 creates the spatial collection to collect all subdomain data. With this arrangement, the solution is rendered and visualized using ParaView in a client-server scheme. The system used for the solution process and rendering is the SGI Altix ICE cluster, with no special graphics hardware. In other words, the visualization is off-screen rendered by the SGI Altix ICE numerical processors using the Mesa library, connected to a ParaView client session. The number of files generated in this XDMF scheme (spatial collection of temporal collections) is proportional to the number of processors and transient output frequency as well and can be computed by the formula $n_f = n_s\left(1+n_p\right)+1$, where n_f, n_s and n_p are the number of files, time steps saved (output frequency) and processors respectively. Consequently, for a run

using 128 processors storing 15 time steps are created a total of 1,936 files. Clearly, for a large number of saved time steps and processor counts, the number of files can be very large. With our solution however, handling large number of files poses no difficulty at all.

4. CONCLUDING REMARKS

In this paper, we discussed the advantages of using an XML-based format to represent data in HPC applications. In this case study heavy information is hierarchically stored in HDF5 files while all information required to access the HDF5 files are stored in XDMF files following the XML standard, thus allowing flexibility, interoperability between different applications, self-description and easy-of-access. However, several other advantages can be foreseen when data is represented in XML. In our case study, XDMF allowed a significant reduction on the number of generated files. Additionally, generating XDMF files from our simulator was trivial. We just added some code, but no real change was needed. This code extension has also been included in other simulators we have. XDMF has all data types to support CFD applications and all our solvers now share the same valid schema. Note though that any other application using VTK can benefit from our solution.

On the other hand, as we generate more XML documents sharing the same schema we can proceed to submit XML queries that would return "all files that have *n* NodesPerElement" or "files use Tetrahedron as topology type". The discussion of proper formats for visualization of volumetric datasets is far from reaching an end. There are several other formats being proposed [6], and no standard yet. Furthermore, depending on the available resources and/or problem size, the end-to-end approach of Tu et al [10], where no intermediate results from simulations are saved, may be preferable.

Acknowledgements. Authors would like to thank CNPq and Petrobras for partially supporting this work.

REFERENCES

1. Clarke, J., Mark, E., Enhancements to the eXtensible Data Model and Format (XDMF). In: DoD High Performance Computing Modernization Program Users Group Conference (HPCMP), pp. 322-327. 2007.
2. Coutinho, A.L.G.A., Martins, M.A.D., Sydenstricker, R.M., Elias, R.N., Performance comparison of data-reordering algorithms for sparse matrix–vector multiplication in edge-based unstructured grid computations. International Journal for Numerical Methods in Engineering, 66:431–460, 2006.
3. Elias, R.N., Martins, M.A.D., Coutinho, A.L.G.A., Parallel edge-based solution of viscoplastic flows with the SUPG/PSPG formulation. Comput. Mech., 38:365-381, 2006.
4. Griebel, M., Dornseifer, T., Neunhoeffer, T., Numerical Simulation in Fluid Dynamics—A Practical Introduction. SIAM: Philadelphia, PA, 1998.
5. Hudec, M. R., Jackson, M. P.A., Terra infirma: Understanding salt tectonics, Earth-Science Reviews 82 (2007) 1–28
6. Kruger, J., Potter, K., MacLeod, R., Johnson, C., UFV – Unified Volume Format: A General System for Efficient handling of Large Volumetric Datasets. In: IADIS Computer Graphics and Visualization 2008 (CGV), 2008.
7. Lins, E.F., Elias, R.N., Guerra, G., Rochinha, F.A., Coutinho, A.L.G.A., Edge-based finite element implementation of the residual-based variational multiscale method. International Journal for Numerical Methods in Fluids, 2009.
8. Moreau, L., Zhao, Y., Foster, I., Voeckler, J., Wilde, M., XDTM: The XML Data Type and Mapping for Specifying Datasets. In: European Grid Conference (EGC), LNCS 3470, 495-505, 2005.
9. Ross, R., Peterka, T., Shen, H.-W., Hong, Y., Ma, K.-L., Yu, H., Moreland, K., Visualization and Parallel I/O at Extreme Scale, Preprint ANL/MCS-P1520-0708, July 2008.
10. Tu, T., Yu, H., Ramirez-Guzman, L., Bielak, J., Ghattas, O., Ma, K-L., O'Hallaron, D., From Mesh Generation to Scientific Visualization: An End-to-End Approach to Parallel Supercomputing. In: SC'06, Tampa, FL, 2006.
11. Valli, A.M.P., Elias, R.N., Carey, G.F., Coutinho, A.L.G.A., PID adaptive control of incremental and arclength continuation in nonlinear applications. International Journal for Numerical Methods in Fluids, 2009.
14. VTK/XML. Available at http://www.vtk.org/VTK/img/file-formats.pdf.

Accelerating Clean Coal Gasifer Designs with Hybrid MPI/OpenMP High Performance Computing

A. Gel[2,1], S. Pannala[3], R. Sankaran[3], C. Guenther[1], M. Syamlal[1], and T. O'Brien[1]

[1]National Energy Technology Laboratory (NETL), Morgantown, WV 26505, USA
[2]ALPEMI Consulting, LLC, Phoenix, AZ 85044, USA
(Tel: 480-763-4745; e-mail: aike@alpemi.com)
[3]Oak Ridge National Laboratory (ORNL), Oak Ridge, TN 37831, USA

Abstract: In this study, we present the impact of high performance computing in the development of advanced environmentally-clean coal-based power generation technologies using MFIX, an open source computational fluid dynamics code. MFIX is a legacy code with over two decades of development towards state-of-the-art models for simulating coal gasification, which was recognized for its uniqueness with a 2007 R&D 100 award. The computational algorithm involves iterative solution of twenty-two nonlinearly coupled conservation equations at each time step. We present how open source scientific software could be used to achieve high-fidelity simulations to address industrial scale problems by reducing the computational time, using high performance computing platforms so that the results can be incorporated in the design cycle. In order to take advantage of the modern high performance computing platforms, a number of improvements were implemented. Preliminary results show that hybrid mode operation can yield substantial improvement in time-to-solution when utilizing thousands of multi-core processors. We expect our experiences and lessons learnt to date would be valuable for other groups using legacy codes.

Keywords: Reactive multiphase flows, industrial scale simulation, fluidization, gasifier model, open source scientific simulation, high performance scientific computing, hybrid MPI and OpenMP parallelization.

1. INTRODUCTION

Environmentally benign energy production is a vital need throughout the world. Meeting the growing demand for clean energy is arguably the most important problem that the world faces today, since the availability of reasonably priced energy is critical to maintaining living standards in the developed world and raising standards of living across the developing world. In the United States, energy demand in recent decades has outpaced the energy supply. Currently, the US imports 60 % of its petroleum, and imports of natural gas are increasing. Moreover, the rising and volatile prices of petroleum and natural gas threaten the economy. The global demand for energy is also rising rapidly, caused by worldwide growth, particularly in developing countries like China and India. The world energy needs are projected to double in the next 30 years and triple by end of the century. Coal is plentiful in the United States, and is currently used to generate more than half of our electric power. Coal-based electric power generation is projected to increase to 156 GW by 2030, which implies the need for significant number of additional full-scale power plants. Coal is a potential source of energy, both in the United States and around the world, not only for future electric power needs, but also as a source of liquid fuels, a substitute for natural gas, a source of chemicals and, possibly, hydrogen in the future.

Advanced technologies must be developed to use coal cleanly, while improving conversion efficiency, and reducing carbon emissions. However, coal-fired power plants concatenate complex unit-operation such as feed systems, gasifiers and combustors of various designs, gas cleanup systems, heat exchangers. These are extremely difficult to interrogate experimentally in detail. Their design is based on experience-based, empirical correlations, which allow only incremental change. Computational tools based on physical principles, such as conservation of mass and momentum, offer an alternative approach to more costly experimental tools. However, current commercially available simulation codes are not accurate enough to be the only design tool, and furthermore, the demand for computational resources to perform realistic simulations in an acceptable timeframe is significant.

Coal gasification is one of the major technologies that are expected to become the centrepiece of tomorrow's green power plants. Coal gasifiers convert coal into a synthesis gas, a mixture of H_2 and CO, which can be cleanly and efficiently converted into power in an Integrated Gasification Combined Cycle (IGCC) process, and the process is amenable for CO_2 sequestration to virtually eliminate the adverse impact of coal on global warming. In addition, the synthesis gas can be converted into transportation fuels or raw materials for chemical or fertilizer manufacture. The U.S. Department of Energy (DOE) is sponsoring a broad research and demonstration program to improve the efficiency, reliability, and feedstock flexibility of gasification systems. To help with the design and troubleshooting of such systems, engineers at the DOE's National Energy Technology Laboratory (NETL) have developed and validated a high-fidelity gasifier model, MFIX.

The insight into the flow field and chemical species and temperature distribution in the gasifier provided by this high-fidelity computational fluid dynamics (CFD) model helps engineers to evaluate different designs and operating scenarios. Usually, such models are used as stand-alone models for evaluating the performance of a single piece of equipment in a unit operation. But CFD models are beginning to be integrated into process simulation models for the entire power plant so that engineers can evaluate the impact of a change in the equipment design on the overall process performance [9]. To ensure that the application of these models is predictive and that they are routinely used in process design, it is crucial that the models are accurate and the time-to-solution cycle time is short.

In this paper, we describe the preliminary results obtained from the improvement of the parallel performance of a coal gasifier model based on an open-source computational fluid dynamics code. A substantial improvement was accomplished by taking advantage of the hybrid MPI and OpenMP mode of operation on a massively parallel multi-core platform, to address the challenge of achieving high fidelity simulations while also reducing the computational time, so that the results can be incorporated in the design cycle.

2. PHYSICAL PROBLEM DESCRIPTION

The transport gasifier, the physical system being modelled in this project, is a vertical tubular reactor (see Figure 1) in which solids (coal and coal-char) and gases undergo chemical changes as they flow upwards. The design of such gas-solids multiphase flow reactors is challenging because of the large spatio-temporal variation of the solids distribution in the reactor. Furthermore, the high temperature and pressure in the gasifier and the opacity caused by the walls in addition to the solids make detailed experimental measurements or visualization of the flow fields in the gasifier virtually impossible. Prior work at NETL has demonstrated that the use of a detailed model can provide useful and sometimes unexpected but highly valuable insights to the designers [5].

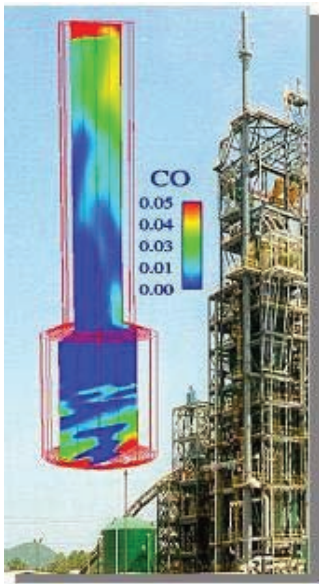


Fig. 1: MFIX simulation of pilot scale transport integrated gasifier at Wilsonville, Alabama, C. Guenther, NETL

3. THE MFIX CODE

MFIX (Multiphase Flow with Interphase eXchanges) is a general-purpose simulation software for modelling the flow dynamics, heat transfer and chemical reactions primarily for gas-solids systems [4], [8]. The state-of-the-art technical significance and uniqueness of MFIX in modelling complex multiphase flows was recently recognized with the 2006 Federal Laboratory Consortium (FLC) Mid-Atlantic Region Tech-transfer award and 2007 R&D 100 award. More recently a multiyear Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program award of DOE was granted to NETL to perform high fidelity simulations of transport gasifier at scales that will be used in commercial plant designs. This tool has been developed primarily at NETL, and released as open-source software in 2001. MFIX has an active international user community of over 1500 users consisting of CFD experts and computational scientists from federal research laboratories, universities and industry, who contribute to the software development [7]. The code is written in Fortran 90 using modern constructs.

MFIX is capable of shared memory and distributed memory parallelization through either OpenMP or MPI in a unified solver framework [1]. It is also capable of leveraging hybrid shared/distributed memory architectures for hybrid parallelism [2]. Shared memory parallelism (SMP) is obtained using portable OpenMP directives. Distributed memory parallelism (DMP) uses flexible and user-determined domain decomposition in all three spatial dimensions. The MPI communication library is encapsulated in a suite of Fortran modules. These modules provide capabilities such as updating the overlapped region, array dot product, and array gather/scatter for I/O operations transparent to the solver modules. Furthermore, the MPI implementation has been optimized over several years by including various techniques such as pre-computing the communication schedule, to allow for packing the messages, minimize any global operations and reusing it as the grid or processor mapping are static. It has a variety of inbuilt iterative linear solvers and can also take advantage of highly scalable high performance linear solver libraries such as Hypre. The code has been ported to various modern high performance computing architectures, ranging from Cray XT4, IBM Blue Gene/P to Linux clusters built with commonly used interconnect networks such as Gigabit, and Infiniband.

A typical gasifier CFD model considers two or more interpenetrating phases and solves conservation equations for mass, momentum, energy, and species mass fractions for each of these phases. The equations describing different phases are tightly coupled because of the interphase exchange of mass, momentum and energy. In addition, the stiffness in the system comes from stiff chemical equations and large variation of the stresses in the solids. MFIX is uniquely suited for simulating multiphase flow problems with a strong coupling between the heavily loaded fluid-particle hydrodynamics, heat transfer and chemical reactions. MFIX provides two approaches for solving transient gas-solids multiphase flow problems encountered in gasifiers: (1) Discrete Element Method (DEM) (also referred to as Eulerian-Lagrangian model); (2) Continuum model (also referred to as Eulerian-Eulerian or Two-fluid Model). The gasifier model presented here is based on the continuum approach, which obviates the need for tracking billions of particles and, therefore, is cost-effective for simulating large-scale, gas-solids flows. In the recent past, MFIX has been used for simulating several challenging multiphase flow problems such as bubbling, spouted and circulating fluidized beds [[6] ,[3]]. More details on the code can be found at the MFIX web site (www.mfix.org).

The strong coupling between the hydrodynamics and complex coal gasification and combustion kinetics are handled in the MFIX gasifier model using the patent pending Carbonaceous Chemistry for Continuum Modelling Module (C3M). The C3M was developed at NETL and received a 2008 FLC National Tech-transfer award. The MFIX gasifier model has been validated by conducting 3D simulations of the Transport Integrated Gasifier (TRIG) from the Power System Development Facility (PSDF) in Wilsonville, Alabama. The PSDF is a DOE demonstration plant for advanced electric-power technologies, and the TRIG is based on circulating fluidized-bed technology that can operate as a coal gasifier. In this plant-sized technology (the reactor unit is 80 feet tall) coal and recycled material feed into the lower part of the gasifier, called the mixing zone, where the coal combusts at high temperature and pressure. Hot gas and unburnt solids rise from the mixing zone into the riser. At the top of the riser, unburnt solids are collected and fed back into the bottom of the mixing zone. Eventually, coal is converted to gas with nearly 100 % efficiency.

4. PERFORMANCE IMPROVEMENTS & PRELIMINARY RESULTS OF HYBRID MODE OPERATION

To improve the performance of MFIX on today's massively parallel multi-core high performance computers such as Cray XT series, various optimizations and improvements were incorporated in several phases. Phase 1 improvements were geared towards optimization without any code change, i.e., determining the best set of compiler flags or MPI environment variables for the batch job. Phase 2 improvements were algorithmic changes, such as

reducing the number of MPI_AllReduce calls in the linear equation solver, reducing residual calculations, etc. Furthermore, the compiler was changed from Portland Group to PathScale during Phase 2 improvements based on independent benchmarks conducted between two compilers.

Phase 3 improvements were based on the extension of a past study to explore parallel performance of MFIX when hybrid shared memory parallel (SMP) and distributed memory parallel (DMP) execution is employed on shared memory multiprocessor systems [2].

Prior to the DMP version of MFIX, an SMP version was developed by manually inserting portable OpenMP directives around the DO-loops in the most time consuming routines in MFIX (approximately 62 locations throughout the code). Due to the limitations in scalability with the SMP mode, the DMP mode of execution, which is based on MPI message-passing library, was routinely used for jobs with large number of processors. In the study of Pannala et al. [2], "one thread per MPI task, one MPI task per processor" was determined to give the best performance with the parallel computer architectures used at that time (e.g., IBM SPs). This observation was consistent with previous hybrid parallelization efforts on somewhat similar architectures. One of the reasons was attributed to the fact that thread creation/destruction is very expensive on the systems employed at that time, which were also single core processor based systems.

The basic idea for this improvement phase was to exploit shared memory parallelism within the cores on the compute node and distributed memory parallelism across the nodes of Cray XT4 platform at National Center for Computational Sciences (NCCS). This approach was not possible on the early Cray XT platforms due to lack of multithread support on the compute nodes. However, with the upgrade to compute node kernel (CNL) and installation of supporting compilers, hybrid mode operation, i.e., MPI and multithreaded OpenMP instructions became a possibility.

Profiling information based on TAU and CrayPAT profiles suggested that apart from the MPI time, significant amount of time is spent in linear equation solver routines, i.e., leq_iksweep and leq_matvec. To isolate the impact of most time consuming routines, out of 62 DO-loop related OpenMP directives that were implemented for SMP mode of operation, only those in the above listed two subroutines were enabled for OpenMP supported compilation and testing on the Cray XT4. Testing for determining the gains for the remaining OpenMP directives is under progress.

Table 1 shows the results obtained from the actual production runs for the 10 million cell resolution transport gasifier model on 2064 cores. A sampling size of eight batch job run sessions were used to obtain the average simulated time per session (last column). The maximum duration for each batch job session was 12 hours of wall clock time due to the queue limitations for the requested number of cores. As seen from the results, employing 1028 MPI processes and 2 threads per process performed nearly 2.5 times better than employing 2064 cores of MPI processes standalone with single thread when the simulated time duration for the gasifier is considered for the same problem, i.e., on average the simulation progressed for 0.18 seconds at the end of 12 hour run for 2064 cores of MPI processes only, whereas the simulation running in hybrid mode progresses 0.47 seconds at the end of the same duration.

Table 1: Comparison of simulated times for MPI only and hybrid MPI + OpenMP mode of operation on Cray XT4 at NCCS

	# of MPI cores	# of threads	Avg. simulated time/12 hr job (sec)
MPI only	2064	1	0.18
MPI + OpenMP	1032	2	0.47

Increasing the number of cores per compute node requires different strategies and algorithmic changes to take advantage of the new high performance computing platforms effectively. Testing is under progress to determine the performance characteristics with higher number of threads and MPI processes. Furthermore, to better understand the observed improvement in performance, additional performance profiling is being performed to compare the profiling results of both modes of operation.

Through these approaches we plan to improve the parallel performance of the code further and reduce the time-to-solution by making the best use of the leadership computing resources. We also plan to extend the MFIX numerical algorithm to allow variable grid size for use with higher order discretizations. This addition will give us the ability to

further improve the resolution and thus accuracy at the locations of interest without increasing the number of grid points significantly.

5. SUMMARY AND CONCLUSIONS

In this study we have illustrated how an open source computational fluid dynamics code could be effectively used in the design of an industrial scale coal gasifier by employing large-scale computational resources. Due to the complexity of the governing physics involved, accuracy and fast turnaround time to solution are the two critical and competing factors in the successful application of computational models to full scale engineering problems. To address both of these key issues MFIX, which has one of the most sophisticated models for multiphase flows, was utilized in hybrid mode by employing domain decomposition through message-passing across the nodes and OpenMP directives based do-loop parallelization within the nodes. This improvement has enabled MFIX to be used for high-fidelity transient simulations that requires over several million computational cells and tens-of-thousands time steps. It is impossible to conduct such simulations on serial machines or even on small clusters because of memory limitations and unacceptably high turnaround time. Such high-fidelity simulations are critical for making substantial improvements in the design of next-generation power plants.

The preliminary results on Cray XT4 supercomputer at NCCS shows that nearly 2.5 fold improvement can be achieved in the simulated time for the same duration of job execution when hybrid MPI and OpenMP directives based multithreading is employed. Further investigation to determine the trade-off between number of threads and number of MPI processes for a given problem configuration and grid resolution is under progress.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of the U.S. Department of Energy, Fossil Energy Advanced Research Program. The submitted manuscript has been authored by several contractors for the U.S. Government under Contract Numbers DE-AM26-04NT41817.670.01.03 and DE-AC05-00OR22725. This technical effort was performed in support of the National Energy Technology Laboratory's on-going research in multiphase flows under the RDS contract DE-AC26-04NT41817. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Also this research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Continuous assistance of Dr. Sameer Shende from University of Oregon for performance profiling is also acknowledged.

REFERENCES

1. D'Azevedo, E. , Pannala, S. , Syamlal, M., Gel, A., Prinkey, M., and O'Brien, T. J., (2001) "Parallelization of MFIX: A Multiphase CFD Code for Modeling Fluidized Beds" presented at *Tenth SIAM Conference on Parallel Processing for Scientific Computing*.
2. Pannala, S., D'Azevedo, E., Syamlal, M., and O'Brien, T. J., (2003) "Hybrid (mixed SMP/DMP) parallelization of MFIX: A multiphase CFD code for modeling fluidized beds," *ACM Symposium on Applied Computing*, Melbourne, Florida USA, March 9-12.
3. Pannala, S., Daw, C. S., Finney, C. E. A., Boyalakuntla, D., O'Brien, T. J. and Syamlal, M. (2007) "Simulating the Dynamics of Spouted Bed Nuclear Fuel Coaters," *Chem. Vapor Deposition*, **13**:481-490.
4. Syamlal, M., (1998) "MFIX documentation: Numerical Technique, Tech. Report, DOE/MC31346-5824 (DE98002029)", Morgantown Energy Technology Center, Morgantown, WV (see http://www.mfix.org).
5. Syamlal, M, (ed.) (2006) "Report on workshop on Multiphase Flow Research," DOE/NETL-2007/1259, Morgantown, WV, June 6-7.
6. Syamlal, M. and O'Brien, T. J., (2003) "Fluid dynamic simulation of O3 decomposition in a bubbling fluidized bed," *AIChE J.*, **49**:2793-2801.
7. Syamlal, M., O'Brien, T. J., Benyahia, S., Gel, A. and Pannala, S., (2008) "Open-Source Software in Computational Research: A Case Study", *Modelling and Simulation in Engineering*, Volume 2008, Article ID 937542, 10 pages, doi:10.1155/2008/937542.
8. Syamlal, M., Rogers, W. A., and O'Brien, T. J., (1993) "MFIX Documentation: Theory guide, Tech. Rep. DOE/METC-94/1004 (DE9400087)" Morgantown Energy Technology Center, Morgantown, WV, (see http://www.mfix.org).
9. Zitney, S. E., Rogers, W. A., Syamlal, M. , Osawe, M. , Madsen, J., and Shi, S., (2005) "Advanced Process Engineering Co-Simulation of Power Generation Systems," *30th International Technical Conference on Coal Utilization & Fuel Systems, Clearwater Coal Conference*, Clearwater, FL, USA, April 17-21.

Report on the development of an generic discontinuous Galerkin framework in .NET

Florian Kummer*

**TU Darmstadt, Fluid Dynamics Group, Hochschulstraße 1, 64283 Darmstadt, Germany*
(email: kummer@fdy.tu-darmstadt.de)

Abstract: In the past $1\frac{1}{2}$ years, the authors have been working on an object-oriented framework for the discontinuous Galerkin (spectral element) method, with a strong aim on CFD applications. This library was programmed in C# for Microsoft .NET and Mono framework. Up to our knowledge, it's the first ambitious CFD code which was implemented using the .NET framework.

Section 1 gives a brief overview of features, library layout and current implementation status. Within the 2nd section, we discuss the advantages which we gain from using a managed platform. The 3rd section compares the performance of C# (and Java) - programs to "classical" languages, discusses performance pitfalls in managed platforms and concludes with some performance measurements on our code. In section 4, we discuss how non-.NET libraries (C or FORTRAN libraries) can be called from C# and we conclude with a parallel scalability study in section 5.

Keywords: Software Frameworks, Parallel Software Development;

1 DESIGN GOALS OF THE LIBRARY

Within the last 10 years, the discontinuous Galerkin method (DG) ([3], [9], [8]), sometimes also referred to as spectral element method, has become very popular for solving hyperbolic conservation laws, especially in computational fluid dynamics ([5], [10]), and is recently strongly emerging in incompressible CFD ([4], [6]).

As our institution is heavily active in combustion and turbulence modeling, modularity, expandability and scalability are central design goals for the library. Hence, the library is designed to model arbitrary systems of the form

$$c_\gamma \cdot \frac{\partial}{\partial t} u_\gamma = \operatorname{div}\left(\mathbf{f}_\gamma(\mathbf{x},U) + \mathbf{L}_\gamma(\mathbf{x})U + \nu(\mathbf{x}) \cdot \nabla G_\gamma U\right) + q_\gamma(\mathbf{x},U) + W_\gamma(\mathbf{x})U, \qquad \gamma = 1,\ldots,\Gamma$$

where $U = (u_1,\ldots,u_\Gamma)$ denotes the vector of unknowns, $\mathbf{x} \in \mathbb{R}^D$ and $c_\gamma \in \{0,1\}$. Actual, the library supports spatial dimensions $D \in \{1,2,3\}$ with both, structured and unstructured grids. Each equation of the system can contain nonlinear fluxes $\mathbf{f}_\gamma$ and sources q_γ, as well as linear fluxes $\mathbf{L}_\gamma$ and sources W_γ. The term $\nu(\mathbf{x}) \cdot \nabla G_\gamma U$ can be used to discretize some linear equations with derivatives of second order (e.g. the Poisson equation) directly, i.e. without breaking them up into a system of first order equations.

The user provides the components of the equation above as well as Riemann solvers for the flux components $\mathbf{f}_\gamma$, $\mathbf{L}_\gamma$ and G_γ by implementing interfaces defined by the library. For the linear components, the code is able to construct sparse matrices automatically from the analytical functions which are provided by the user.

Additionally, there is a parallel database system for storing, loading and *organizing* the data which is produced by the numerical methods. Of course, the code is MPI parallelized, and it is capable of handling arbitrary repartitioning of the grid (load balancing), which can come e.g. from external libraries like ParMETIS ([12]). Large parts of the quadrature kernel are vectorized in a way that can be described as "multiple instruction - multiple data". We are able to run the code on Linux and Windows machines/clusters.

Currently, we are able to get first results of the full incompressible Navier-Stokes - solver.

2 ADVANTAGES OF MANAGED PLATFORMS

The major difference between so-called *managed languages* like Java or C# and "classical" programming languages is that the first ones are compiled to use a runtime Environment (the .NET framework for C# or the Java runtime)

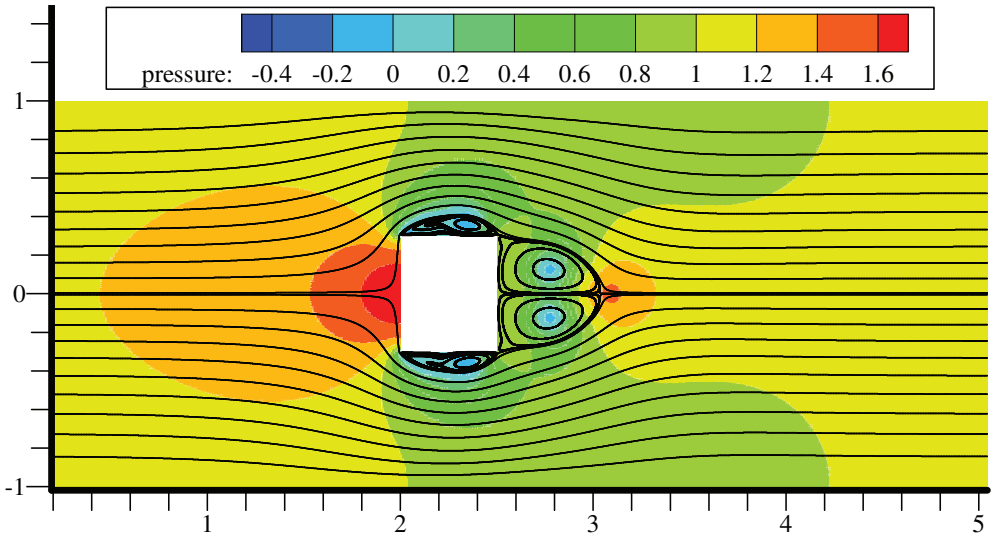


Figure 1: *Incompressible Navier Stokes, Channel Flow around obstacle; Grid of* 874×279 *cells, Reynolds number 1000; Pressure distribution and streamtraces; Timestep No. 2331, Physical Time* $t = 1.289$*;*

which does not only deliver an extensive runtime library but also runtime services like *garbage collection*, *runtime-type-information* and *serialization*.

Usually, for software projects which are done on Universities or similar institutions, manpower is the most limited resource. We have found that the usage of a managed language together with a comprehensive toolchain (an Integrated Development Environment) is speeding up our workflow compared to the "classical" development environment (text editor, shell, makefile).

One thing found very nice with our code is the fact that it is platform independent on a binary level. This means that once we have the .NET runtime (on Unix systems the compatible Mono runtime [14]) and some native libraries (e.g. BLAS, MPI, LAPACK, HYPRE [13]) installed on some cluster or supercomputer,we can directly copy and run the executables from our local development workstations.

We further also use some essential technologies of .NET, which cannot be found in classical languages: Serialization is the process of packing an arbitrary graph of objects into an continuous block of memory, which can be streamed either to hard disk or via MPI to another process. As with multi-purpose codes data structures may get complex, serialization proofed to be a big help, for File IO and parallelization and general interprocess communication.

A wide range of programming errors (e.g. buffer overflow) that usually occur during FORTRAN of C development, are prevented by the design of the .NET runtime. Memory leaks like in C, when allocated heap memory isn't freed cannot occur in .NET. A so-called *garbage collector* keeps track of all heap objects, and frees them if they aren't referenced by the application anymore and if memory is needed. Furthermore, the garbage collector is also capable of compacting the heap.

3 MANAGED PLATFORMS AND PERFORMANCE

It is a popular fallacy that C# (or Java) is not suitable for High Performance Computing, because it is an interpreted language. This is not the case. The output of the C#-compiler is so-called *Common Intermediate Language* (CLI), looking much like an assembler program plus metadata to represent object oriented structures. At load time, this CLI code is transformed to native-machine code.

Since the first JIT-versions of Java have been available, various authors have made performance comparisons between C and Java (e.g. [7] and [2], just to mention two of them). These investigations concluded that Java, when properly used, is suitable for high performance computing. When .NET was released, it's performance was and is continuously compared to Java (e.g. [1]). C#, in contrast to Java, further has the option to declare specific regions of the code

as *unsafe*. Within these sections, runtime security features (like array bounds - checking) are turned of, and pointer arithmetics like in C is allowed.

The - by our knowledge - most comprehensive benchmarks (including LINPACK and SciMark) were done in [11]. We also performed some benchmarks on basic algorithms (naive DGEMM and the Sieve of Eratosthenes).

	DGEMM, $N = 1000$	Sieve of Eratosthenes, $p < 2 \cdot 10^7$
C#, Debug, .NET 3.5	29.8 sec	39.6 sec
C#, Release, .NET 3.5	16.7 sec	37.6 sec
C#, Release, Mono 2.2	34.9 sec	38.9 sec
C#, unsafe code, .NET 3.5	9.8 sec	37.7 sec
C#, unsafe code, Mono 2.2	12.4 sec	39.5 sec
C, MinGW-gcc 3.4.5	16.8 sec	30.3 sec
C, MinGW-gcc 3.4.5 -O3	9.8 sec	20.9 sec
C, MS-cl 15.00.21022.08	15.5 sec	26.8 sec
C, MS-cl 15.00.21022.08 /O2	9.7 sec	37.6 sec
ACML-BLAS 4.1.0	1.28 sec	n.a.

System: Pentium 4 (Prescott), 3GHz, Windows XP service pack 3

From our own test and benchmarks done by others we reason that...

- Optimized binary libraries from hardware vendors are usually superior to naive implementations build by any compiler.
- LINPACK and SciMark show that classical languages are still faster, but not by much - especially for the LINPACK case (see [11]).
- The performance loss of using a managed language in the worst cases is about 50%. Wether that really happens in real-world applications is uncertain.
- Overall, Java and .NET provide equal performances. On some algorithms, there may be a big gap.
- It's difficult to find proper benchmarks, because more complex algorithms cannot be translated one-by-one from C# to C or vice versa.
- Unsafe code performs very often head-to-head with state-of-the-art C compilers.
- Especially Mono seems to profit a lot from using unsafe code.
- Although we may expect a performance loss in the range of 20 to 40%, the advantages in development outbalance the performance loss.
- Data, on which performance-critical algorithms work on, should be organized in arrays of value types. Complex graphs of heap objects should be avoided.
- Performance-critical sections should be "vectorized", i.e. implemented as multiple instruction-multiple data. Loops should be preferred to multiple function calls, i.e. the stack should be kept shallow. This opens the possibility to optimize the code with unsafe sections.

4 ON THE USAGE OF EXTERNAL LIBRARIES

From our point of view, there are three reasons for using native code in .NET applications for supercomputers: First, the need for near-system libraries like MPI, which are not available (natively) in the .NET runtime library. Second, the use of complex mathematical libraries like sparse solvers, which would be too much work to re-implement them in C#. And third, the need for performance. As shown above, a DGEMM operation from an optimized library is superior by nearly a magnitude of 10 to naive implementations in *any* programming language.

The standart way of using native code in C# is called *P/Invoke*. Native code is compiled/linked into a shared library (.dll - files in Windows, .so - file on Unix) and within the C#-code, a function prototype together with an attribute that points to the shared library, is defined. P/Invoke *marshalles* the managed objects to the unmanaged code. The most important thing is to prevent the garbage collector (which may run at any time in it's own thread) from moving the base address of some object while an unmanaged function accesses it. This "looking" of objects for the garbage

collector is referred to as *pinning*. For standart function calls, which take some input data and process some output, pinning is performed automatically by P/Invoke.

Besides that, there are some function calls that require that pinning continues for some time after they return - e.g. nonblocking MPI calls like MPI_Irecv. In .NET, it's relatively easy to control pinning manually, which isn't possible in Java.

5 PARALLEL PERFORMANCE AND SCALING

Parallel speedup was measured on a benchmark which simulates scalar advection in a given flow field in 3D. Two experiments were made, one with constant problem size and one with growing problem size. Except for the 1- and 2-processor tests, all processes but "first" and "last" one have to communicate with two neighbors. Therefore, the runtime of the 4-processor-test was taken as the basis to calculate the speedup of the other tests.

The test was carried out on a 64-processor cluster, organized in 8 nodes with two quad core Opteron processors at 2 Ghz and Gigabit Ethernet interconnect.

# of Procs	Runtime [sec]	# of Nodes	Speedup	Parallel Efficiency
2	36371	1	2.03	1.01
4	18422	1	4	1
8	9877	1	7.46	0.93
12	6606	2	11.15	0.93
16	4897	3	15.05	0.94
24	3318	4	22.21	0.93
32	2559	5	28.8	0.9
48	1765	7	41.75	0.87

Parallel speedup of a benchmark with constant problem size. Grid is $960 \times 9 \times 19$ *cells.*

As expected, the speedup of a test with constant problem size is limited, because the ratio between communication and calculation gets worse with increasing processors. The number of cells to exchange stays constant (342 cells to send and receive for each process) and e.g. for the 4-processor test, there are 41040 cells on each processor to calculate. For 48 processors, the number of cells per processor drops down to 3420.

# of Procs	Runtime [sec]	# of Nodes	Speedup	Parallel Efficiency
1	1496	1	1.15	1.149
2	1693	1	2.03	1.015
4	1719	1	4.00	1
8	1759	2	7.82	0.977
12	1725	2	11.96	0.997
16	1742	3	15.79	0.987
24	1787	4	3.09	0.962
32	1788	5	30.77	0.961
48	1786	7	46.20	0.962
56	1776	8	54.20	0.968

Parallel speedup of a benchmark with growing problem size. Grid is $20 \cdot n_{Proc} \times 9 \times 19$ *cells. (Speedup should be understood as runtime in comparison to the extrapolated runtime form the 4-processor test.)*

Within the test above, we see a significant performance drop between 8 and 24 processors, beyond that point the parallel efficiency seems to remain constant. Wether this is a result of hardware "saturation" (e.g. the network switch) or results from the algorithm has to be determined by test on clusters with better interconnect and more processors. It is clear, that for 1 processor, where communication is turned off, the speedup in comparison to 4 processors is greater than 1.0.

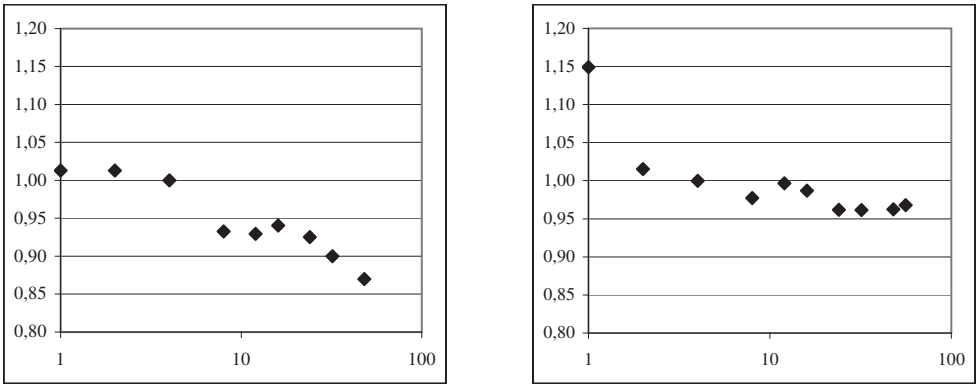


Figure 2: *Parallel efficiency versus number of processors. (left side: constant size problem, right side: growing size problem)*

REFERENCES

[1] W.Vogels (2003). Benchmarking the CLI for high performance computing. *IEE Proc.-Softw.*, Vol. 150, No. 5, October 2003;

[2] J.M. Bull, L.A. Smith, L. Pottage, R. Freeman (2001). Benchmarking Java against C and Fortran for scientific applications. *Proc. ACM Conf. on Java Grande/(ISCOPE)*, Stanford University, CA, 2-4 June 2001,pp. 97-105;

[3] D.N. Arnold, F. Brezzi, B. Cockburn, L.D. Marini (2002). Unified analysis of dincontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, Vol. 39, No.5, pp. 1749-1779;

[4] K. Shahbazi, P.F. Fischer, C.R. Ethier (2007). A high-order discontinuous Galerkin method for the unsteady incompressible Navier-Stokes equations. *J. Comp. Phys.* 222 pp. 391-407;

[5] F. Bassi, A. Ghidoni, S. Rebay, P. Tesini (2008) High-order accurate p-multigrid discontinuous Galerkin solution of the Euler equations. *Int. J. Numer. Meth. Fluids*, Early View;

[6] V. Griault, B. Riviere, M.F. Wheeler (2005) A splitting method using discontinuous galerkin for the transient incompressible Navier-Stokes equations. *ESIAM: M2AN* Vol. 39, No. 6, pp. 1115-1147

[7] Java Grande Forum Report: Making Java work for high-end computing. Java Grande Forum Panel; Presented at Supercomputing 1998, 13 November 1998, http://www.javagrande.org/reports.htm;

[8] J.S. Hesthaven, T. Warburton (2008) *Nodal Discontinuous Galerkin Methods - Algorithms, Analysis and Applications*, Springer-Verlag, ISBN 978-0-387-72065-4

[9] B. Cockburn, G.E. Karniadakis, C.W. Shu (2000); *Discontinuous Galerkin Methods*, Springer-Verlag; ISBN 3-540-66787-3

[10] V. Dolejsi, M. Feistauer (2004). A semi-implicit discontinuous Galerkin finite element method for the numerical solution of inviscid compressible flow. *J. Comp. Phys.* 198 pp. 727-746

[11] Kazuyuki Shudo (2005). Performance Comparison of Java/.NET Runtimes. http://www.shudo.net/jit/perf;

[12] G. Karypis, V. Kumar (1996) Parallel multilevel k-way partitioning scheme for irregular graphs. Technical Report TR 96-036, Department of Computer Science, University of Minnesota, 1996

[13] HYPRE - high performance preconditioners. http://acts.nersc.gov/hypre

[14] http://www.mono-project.com

LARGE-SCALE APPLICATION SCALING

Scaling Applications to 100,000 Cores and Beyond on IBM Systems

Jeff Fier*, Jeff Zais**

**IBM Corporation, Rancho Mirage, CA 92270, USA*
(Tel: 760-651-5005; e-mail: jfier@us.ibm.com)
***IBM Corporation, Hudson, WI 54016, USA*
(e-mail: zais@us.ibm.com)

Abstract: Breakthoughs in science will require ever increasing computer performance. But lack of efficient programming models and physical constraints such as power consumption are challenges that must be overcome to achieve petascale computing and beyond. IBM is addressing these challenges with a multi-dimensional approach, exemplified by the high-productivity Blue Waters system, the energy-efficient Blue Gene, and hybrid computing systems such as the "Roadrunner" project for Los Alamos National Laboratory.

Keywords: Blue Waters, Blue Gene, petascale computing, hybrid computing.

1. INTRODUCTION

Petascale computing holds the promise of breakthroughs in all fields of science and engineering[7]: aircraft manufacturers will be able to better model full aircraft and create more fuel efficient and quiet designs; biologists will be able to better simulate protein structure, leading to advances in disease control; the automobile industry will be able to explore the detailed chemistry and physics of turbulent flames, allowing improved engine design; weather and climate simulations will be able to use high enough resolution and more detailed models of chemical and physical processes to allow reliable predictions on a regional scale. But many challenges lie on the road to and beyond sustained petascale computing, among them: supplying and paying for the power required to run petascale computers and finding efficient ways to program systems that have 100,000 and more cores.

Several approaches are evolving to address these challenges, although they are sometimes at odds with each other. One way to address power efficiency is to construct systems from smaller, relatively slower processors [1]. The lower clock speed greatly reduces power consumption, but many more processors are required to achieve a particular level of performance. Alternatively, one may employ accelerators such as the PowerXCell 8i [6] or general purpose graphical processing units (GPGPUs) along with a general purpose microprocessor [4]. Since accelerators only contain hardware for specialized processing, they can achieve performance per watt figures an order of magnitude better than general purpose processors. But constructing systems from homogeneous cores with high single-thread performance tends to make the programming challenges less onerous since it preserves familiar programming models and minimizes the cost of sequential code sections. As a result there may be no single approach that is best in all situations.

IBM has long been a leader in high performance computing (HPC) and several announcements in the past year exemplify the multi-dimensional approach IBM is taking to addressing the challenges to petascale computing:

- The Los Alamos National Laboratory (LANL) "Roadrunner" system: the first machine to achieve one petaflop
- The Blue Waters High Productivity Computing System
- The "Dawn" and "Sequoia" systems for Advanced Simulation and Computing (ASC) at Lawrence Livermore National Laboratory (LLNL)

These systems employ different approaches to address the challenges facing petascale computing. Dawn and Sequoia, like their predecessor Blue Gene/L, use slower, homogeneous cores, but achieve high system performance with the highest number of cores. Roadrunner uses heterogeneous cores, PowerXCell 8i accelerators, and multiple, hybrid programming models to maximize flops per watt. Blue Waters emphasizes high productivity computing with homogeneous cores and maximum single core performance.

Scaling applications to take full advantage of these systems is a challenge. But early experience has shown that it is possible and exceptional results have been achieved. In the sections that follow, we will describe each of these systems and some of the approaches used in scaling applications on them.

2. BLUE GENE/L, DAWN, SEQUOIA

In February of this year, The DOE's National Nuclear Security Administration (NNSA) announced that IBM had been selected to design and build two new supercomputers to continue the work of the NNSA's Stockpile Stewardship program currently carried out on the ASC Purple and Blue Gene/L systems at LLNL. Dawn, the first of the two systems, is a 500 teraflop system based on IBM's Blue Gene/P technology and is scheduled for delivery early this year. Sequoia, which will be based on future IBM technology, will exceed 20 petaflops of performance, employ 98,304 nodes in 96 racks with 1.6 terabytes of memory, and put 1.6 million cores to work in unison. It is scheduled for delivery in 2011, with operational deployment in 2012.

Scaling applications on such systems will no doubt be a challenge. But their architecture, based on many low-power, homogeneous cores, has the advantage of being able to build on the experience and successes achieved on their predecessor, Blue Gene/L [1]. Using that system—which at the time contained 128k cores and had a peak speed of 367 teraflops—scientists at LLNL garnered Gordon Bell peak performance prizes in 2005 and 2006. And with the system expanded to 208k cores, anther LLNL team won the 2007 Gordon Bell prize [3].

In the first of these efforts, the molecular dynamics code ddcMD was used to simulate solidification in molten tantalum and quenched uranium, achieving a sustained performance of 101.7 teraflops [8,9]. In order to scale efficiently on massively parallel systems such as Blue Gene/L, ddcMD was designed to use a particle-based decomposition rather than the spatial decomposition traditionally used in MD codes. As a result of this novel approach, simulations were able to scale to the full machine with less than 1% of the run time spent in communications. Load imbalance at 128k cores was measured at 7%. Although not enough to significantly impact scaling at this processor count, it did suggest that adding dynamic load balancing to the code would be required to scale to significantly larger systems. Other keys to scalability were managing data files intelligently (one file per task or one file per job do not perform well due to metadata and bandwidth issues) and making sure that processor and/or domain logic did not scale badly. (We note that in other ports to Blue Gene/L, we have observed $O(N^3)$ algorithms lurking in codes that had previously been observed to scale well on large Linux clusters.)

For the 2006 Gordon Bell award, Gygi et al. [5] used the first principles molecular dynamics code Qbox to simulate high-Z metallic systems, achieving 207 teraflops. Qbox was designed to scale well on massively parallel systems. It maintains good load balance via an efficient data layout and careful management of data flow. Its calculations, however, are not a natural fit for the 3D torus interconnection network on Blue Gene/L. To optimize performance the Qbox team carefully studied the code's communications patterns and found that the best performance was achieved with either a bipartite (when running on subsets of the machine) or quadpartite (for the full machine) mapping of tasks to Blue Gene/L cores. The difference in going from the default mapping to the best mapping improved the code's performance by a factor of 1.67.

Other outstanding results have been obtained for a wide variety or applications on Blue Gene/L [2]. As above, they found that keys to good scaling included proper mapping of tasks to the cores, and optimizing interprocessor communication, particularly when involving all-to-all operations.

3. ROADRUNNER

Built by IBM for the NNSA and LANL, Roadrunner is the world's first hybrid supercomputer[11]. It consists of 6,912 dual-core Opteron processors with 12,960 PowerXCell 8i accelerators and an InfiniBand network. It made worldwide headlines in June 2008 when it became the first computer to break the 1 petaflop barrier. The Opteron processors alone would make this a formidable Linux cluster with a peak processing rate of 50 teraflops. But it is the PowerXCell 8i processors, with a combined peak of 1.3 petaflops that add real power to the system. And that power is efficient as Roadrunner is near the top of the Green500's list of most energy efficient supercomputers.

Each PowerXCell 8i processor consists of 9 cores: a Power Processing Element (PPE) that runs Linux, and 8 Synergistic Processing Elements (SPEs) to carry out number-crunching. In total, there are over 130,000 cores in the system. Running applications efficiently on any system with that many cores is going to be a challenge. But with a hybrid architecture (and even the Cell processors have a hybrid architecture), how do you run applications on this machine?

There are several programming models used on Roadrunner [11]: A host-centric model starts with the application split across the hosts (Opterons) like any other Linux cluster. The PowerXCell 8is are then used to offload and accelerate portions of the work. This was the model initially used in the port of SPaSM (*S*calable *Pa*rallel *S*hort-range *M*olecular Dynamics) [10]. This is an attractive programming model in that it allows an evolutionary approach to porting. In the case of SPaSM, the force calculation was offloaded to the Cell processors, leaving the rest of the calculations and MPI communications on the hosts. This worked well enough, achieving 100 teraflops. But by redesigning the code to run mainly on the Cells, with the Opterons handling only the message passing, the code was converted to an accelerator-centric programming model. That change improved the performance by more than a factor of three to 369 teraflops. Other programming model variants are possible as well. For example, an asynchronous "work-stealing" version of the host-centric model has been used for the Monte Carlo code Milagro.

The unifying theme, though, is that for best performance, fully understanding the code and being willing to redesign it to efficiently use the computing resources is recommended to achieve the best performance. In a tutorial on moving scientific simulation codes to Roadrunner [12], Woodward et al. advocate converting codes to a stream processing paradigm and avoiding global communication.

4. BLUE WATERS

Blue Waters [7] is a collaboration between the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications (NCSA), IBM, and the Great Lakes Consortium for Petascale Computing (GLCPC), funded by a grant from the National Science Foundation, to build a sustained petaflop computational system dedicated to open scientific research.

The system is based on the IBM POWER7 processor and will contain 200,000+ cores. It will have 1 petabyte of globally addressable main memory, 10 petabytes of disk storage, half an exebyte of archival storage and up to 400 Gbps external connectivity. The goal of sustaining a petaflop on applications spanning a wide range of scientific disciplines led to the design choices to maximize single core performance, in order to keep the total core count down and minimize the cost of poorly scaling code sections, and provide a large, global, high-bandwidth memory system to efficiently solve memory-intensive problems.

While fast cores, memory and interconnect hardware will help with some of the challenges in scaling applications to a machine of this size, issues with algorithmic scalability, load imbalance, etc. will be addressed via three interconnected efforts: 1) The Consulting Office will provide consulting services to help researchers port and tune their codes for Blue Waters. 2) Petascale Application Collaboration Teams (PACTs) will be assembled to work closely with developers of specific applications to optimize them and resolve problems in scaling them to petaflop performance levels. 3) Application Development Workshops will be held twice a year to train researchers how to develop codes that will run efficiently on Blue Waters. In addition, software projects at UIUC and IBM to produce tools for developing scalable code on Blue Waters will be tightly coupled with these three efforts.

REFERENCES

1. Adiga, N.R. et al. (2002). An Overview of the Blue Gene/L Supercomputer. *SC2002*. Baltimore, MD.
2. Almasi, G. et al. (2005) Scaling Physics and Material Science Applications on a Massively Parallel Blue Gene/L System. *ICS05*. Cambridge, MA.
3. Glosli, J. et al. (2007). Extending Stability Beyond CPU Millenium: A Micron-Scale Atomistic Simulation of Kelvin-Helmholtz Instability. *SC2007*. Reno, NV.
4. Gschwind, M. et al. (2006). Synergistic Processing in Cell's Multicore Architecture. *Hot Chips 17*. Stanford University, Palo Alto, CA.
5. Gygi, F. et al. (2006). Large-Scale Electronic Structure Calculations of High-Z Metals on the Blue Gene/L Platform. *SC2006*. Tampa, FL.

6. IBM (2008). PowerXCell 8i Processor. *http://www-03.ibm.com/technology/resources/technology_cell_pdf_PowerXCell_PB_7May2008_pub.pdf*
7. NCSA (2009) Blue Water Project, Sustained Petascale Computing. *http://www.ncsa.uiuc.edu/BlueWaters/bw-booklet.pdf*
8. Streitz, F. et al. (2005). 100+ Tflop/s Solidification Simulations on Blue Gene/L. *SC2005*. Seattle, WA.
9. Streitz, F. (2006). Simulating Solidification in Metals at High Pressure. *SDSC Summer Institute*. SDSC, San Diego, CA.
10. Swaminarayan, S. et al. (2008) 369 Tflop/s Molecular Dynamics Simulations on the Roadrunner General-Purpose Heterogeneous Supercomputer. *SC2008*. Austin, TX.
11. Turner, J. (2008) The Los Alamos Roadrunner Petascale Hybrid Supercomputer Overview of Applications, Results, and Programming. *Roadrunner Technical Seminar Series*. LANL Technical Report LA-UR-08-2412.
12. Woodward, P. et al. (2008) Programming Techniques for Moving Scientific Simulation Codes to Roadrunner, *http://www.lanl.gov/orgs/hpc/roadrunner/pdfs/Woodward.pdf*

Performance of CFD Applications on NASA Supercomputers

Jahed Djomehri [*], Dennis Jespersen, James Taft [**], Henry Jin, Robert Hood [*], Piyush Mehrotra

NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center, Moffett Field, CA 94035, USA
{Jahed.Djomehri, Dennis.Jespersen, James.Taft, Haoqiang.Jin, Robert.Hood, Piyush.Mehrotra}@nasa.gov

Abstract: We study the performance of three CFD applications—CART3D, OVERFLOW, and USM3D—on two quad-core based SGI Altix ICE supercomputers and compare with their performance on an SGI Altix 4700 system. We find that the quad-core systems pose significant challenges in terms of cache size and memory bandwidth. However, when we compare performance in a way that attempts to compensate for large differences in cost, the quad-core systems can be the most cost-effective.

Keywords: Multi-core processors, high-performance computing, parallel performance.

1. INTRODUCTION

One strategy used in the design of high-end computers is to take advantage of economies of scale by using commodity parts such as microprocessors. While resulting systems typically have very high peak performance compared to other machines in the same price range, they pose challenges for programmers. In particular, improvements at the core level have not been matched by gains in the performance of the memory hierarchy and interconnect. Furthermore, these components are shared among several cores, leading to contention issues, especially with many CFD codes that stress the memory hierarchy. In this paper we investigate how commodity component architecture affects performance in CFD codes. In particular, we investigate the performance of three codes on three high-end computing systems available at the NASA Advanced Supercomputing (NAS) facility at the Ames Research Center. Two of the systems are SGI Altix ICE systems based on Intel Xeon quad-core processors. The third system is an SGI Altix 4700 shared memory system that uses Intel Itanium dual-core processors. Our three CFD applications are: CART3D, a code solving the inviscid flow equations using adaptively refined Cartesian grids; OVERFLOW, a Reynolds-averaged Navier-Stokes solver employing overlapping structured grids; and USM3D, a Reynolds-averaged Navier-Stokes solver based on unstructured tetrahedral grids. We begin with a description of the target platforms and then discuss the performance of the codes on these architectures.

2. EXPERIMENTAL ENVIRONMENT

In this section, we describe the experimental environment for this study including a brief overview of the three systems used in this study. Further details of the systems can be found at the NAS website [1].

Two of the parallel systems used in this study are part of the Pleiades supercomputer, an SGI Altix ICE system located at NASA Ames Research Center. Pleiades comprises 6400 nodes interconnected with InfiniBand in a hypercube topology. All but 512 nodes contain two quad-core Intel Xeon E5472 processors (Harpertown, 3.0 GHz), while the remaining nodes (also known as RTJones) use two quad-core Intel XeonX5355 processors (Clovertown, 2.66 GHz). This makes a total of 51,200 cores in the whole system. A block diagram of a Pleiades node is shown in Figure 1(a). Each Harpertown and Clovertown processor has a pair of L2 caches that are each shared by two cores; the two processor types also have a similar bus-based architecture. Compared to Clovertown, Harpertown has a faster clock speed, a larger L2 cache, and a faster front-side bus (FSB) speed. Both Pleiades and RTJones utilize InfiniBand in a hypercube topology to interconnect the nodes. However, Pleiades uses newer technology and hence has higher bandwidth and lower latency inter-node communication. In most Pleiades nodes, the eight cores share 8GB of memory. However, 64 nodes (512 cores) have double the memory, sharing 16 GB across 8 cores.

[*] Employee of Computer Science Corporation ** Employee of Sienna Software, Inc.

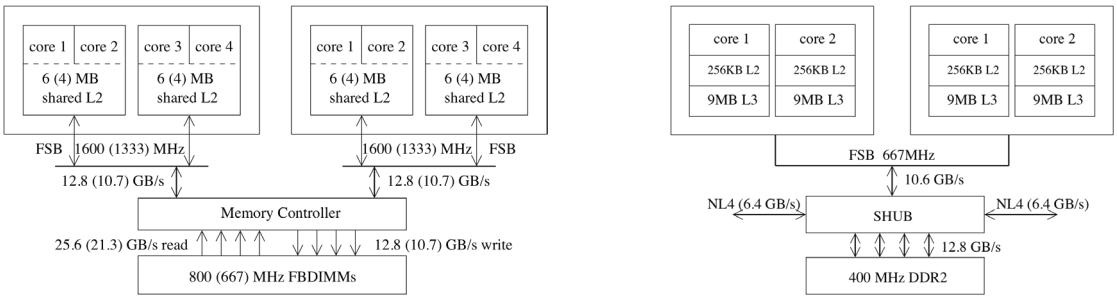


Figure 1: (a) Block diagram of Pleiades (RTJones) node with two Harpertown (Clovertown) processors
(b) Block diagram of Columbia C24 blade with two Itanium2 Montvale processors

The Columbia system is a cluster of SGI Altix nodes interconnected via NUMALink4 (NL4) and InfiniBand. In this paper, we focus on one node of Columbia: C24, an SGI Altix 4700 with 512 Intel dual-core Itanium2 Montvale chips at 1.67 GHz. Each of the 1024 cores has access to 256KB of L2 data cache, 9 MB of L3 cache and 2 TB of globally addressable memory. The system uses SGI's NUMALink4 technology to interconnect the cores. A block diagram of a C24 blade is shown in Figure 1(b).

On the ICE systems, the eight cores share resources including caches, memory, memory bandwidth and also access to the interconnection network. On these systems, the user can choose the number of active cores per chip to be used in a run. Leaving some cores idle reduces memory contention for the remaining active cores, as well as avoiding cache sharing with neighboring cores. This feature allows the user in effect to "dial" in the performance on the cluster to optimize time to solution or minimize cost by running somewhat longer on fewer cores.

In the rest of the paper, we designate runs as "*N*ppn" when we are using *N* processes per node. Thus, 8ppn is a fully populated run while 4ppn and 2ppn leave 4 and 6 cores idle respectively. In the latter two configurations, we have utilized a core configuration that minimizes sharing of resources. Specifically, in 4ppn mode the processes were placed so that each process had exclusive access to an L2 cache (i.e., no cache sharing), and in 2ppn mode that processes were placed so that each process had exclusive access to an L2 cache and a pathway to memory (i.e., no cache sharing and no bandwidth sharing). Note that to run a *P*-process job in 4ppn mode requires double the number of nodes required in 8ppn mode.

All applications discussed in this paper were compiled using the Intel compiler and the SGI message passing library, MPT. The MPT library has many environment variables. Two of these, MPI_BUFS_PER_HOST and MPI_BUFS_PER_PROC, control the number of buffers used for inter- and intra-node communication. Larger values of these variables can improve communication speed by reducing time spent waiting for buffers. However, extra memory for the MPI library can result in insufficient memory for the application, and balancing memory for the MPI library and for the application has an impact on application performance as discussed in the sections below.

In the next three sections, we detail the performance of three CFD codes—CART3D, OVERFLOW, and USM3D—on the three systems described. In each section, we start with a description of the code and the test dataset being run. We then present the raw timings for the configurations of interest on each platform. Finally, each section then analyzes how the different architectures affect the performance of that particular code. As part of this analysis, we attempt to account for large cost differences among the systems. In particular, we present graphs where application performance is compared on a socket-per-socket—rather than core-per-core—basis. To a rough approximation, this levels the playing field by comparing the aggregate performance of a quad-core Xeon processor to that of a dual-core Itanium processor. The graphs plot the time per socket for runs on Pleiades and RTJones relative to that on C24 versus the number of sockets used - a value greater than one implies C24 performs better than the other system.

3. CART3D

CART3D is a simulation package targeted at conceptual and preliminary design of aerospace vehicles with complex geometry [2]. The flow simulation module solves the Euler equations governing inviscid flow of a compressible fluid. CART3D's solver module uses a second-order cell-centered, finite-volume upwind spatial discretization combined with a multigrid accelerated Runge-Kutta scheme for advance to steady-state. CART3D uses a variety of techniques to enhance its efficiency on distributed parallel machines. It uses multigrid for convergence acceleration and employs a domain-decomposition strategy for subdividing the global solution of the governing equations among the many processors of a parallel machine.

CART3D takes advantage of the hierarchical nesting of adaptively refined Cartesian meshes and uses techniques based upon a Space-Filling-Curve (SFC) for reordering of the adapted meshes. The locality properties of the SFC ordering are such that a good partitioning strategy is to simply distribute different segments of the SFC among the various processors. All meshes in the hierarchy are partitioned independently using the same SFC. This implies that although there will be generally very good overlap between corresponding fine and coarse partitions, they are not perfectly nested. While most of the communication for multigrid restriction and prolongation in a particular subdomain will take place within the same local memory, these operators will incur some degree of off-processor communication as well. This approach favors balancing the workload on each mesh in the hierarchy at the possible expense of increased communication.

For large datasets, CART3D is CPU and memory bound. Due to the underlying unstructured mesh and the resultant indirect addressing, the code is inherently less cache friendly than codes utilizing structured meshes, however the use of SFC greatly improves data locality. The test case used in this study, SSLV, consists of 24M cells and models a launch configuration of the Space Shuttle.

Table 1: CART3D Time per cycle (secs)

No. of Processes	C24	RTJones		Pleiades			
		8 GB/node		*8 GB/node*		*16 GB/node*	
		8ppn	*4ppn*	*8ppn*	*4ppn*	*8ppn*	*4ppn*
32	18.20	40.89	26.56		21.28	14.87	10.11
64	9.03	20.08	13.05	14.92	9.98	8.14	5.44
128	4.80	8.49	6.20	6.21	4.68	4.20	2.78
256	2.46		3.01		2.19	2.13	1.38
512	1.36		1.73			1.03	

In Table 1, we compare the time in seconds required per cycle of the code across the different architectures; this excludes initialization and I/O time. The blank spaces in the table denote runs that could not be performed due to insufficient memory to execute the code. CART3D scales well on all the machines--even superlinearly in many cases. This is due to increased total cache and memory bandwidth as processor counts go up. Cache size also explains how a slower-clocked C24 can often outperform the ICE platforms.

The runs on the big memory nodes (16 GB/node) of Pleiades scale nearly linearly and performance is more than 30% better than those on the small memory nodes of Pleiades. This is due to allocating more memory for the MPI library for buffering (through the environment variables MPI_BUFS_PER_PROC and MPI_BUFS_PER_HOST). Although we don't include the numbers in Table 1, if we use the extra memory on the 4ppn runs on the 8GB nodes for MPI buffers, we see similar timings to those for 4ppn runs on 16 GB nodes. The runs on the larger memory nodes of Pleiades were uniformly faster than those on C24 even on a per-core basis.

Comparing the 8ppn to the 4ppn results on Pleiades shows that there can be a significant boost in performance by leaving every other core idle (using the extra memory for the MPI library). The 4ppn-16GB/node runs on Pleiades are about 1.5 times faster than the 8ppn runs. There seem to be four advantages when 4ppn is used: cache per active core is doubled, memory bandwidth is doubled, the ratio of ports to cores is also doubled decreasing the congestion for inter-node communication, and the memory for the MPI library can be increased.

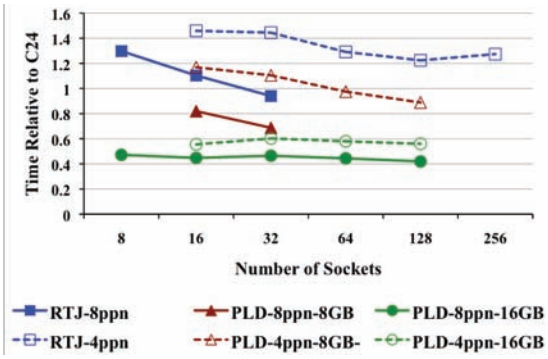


Figure 2: Relative performance of CART3D

Figure 2 shows scaling plots of CART3D on the three parallel systems as the number of sockets used is increased. As seen in the figure, C24 outperforms 4ppn runs on RTJones for the whole range of sockets, and C24 outperforms some of the RTJones 8ppn and Pleiades 4ppn cases for smaller numbers of sockets. However, all the Pleiades cases with 8ppn, all the Pleiades cases with 4ppn-16GB/node, and the Pleiades 4ppn-8GB/node cases with 64 and 128 sockets outperform C24. The 4ppn-16GB/node and 8ppn-16GB/node runs on Pleiades are nearly 1.6x and 2x faster than on C24, respectively.

4. OVERFLOW

OVERFLOW is a general-purpose Navier-Stokes solver for CFD problems [3]. The code uses an overset grid methodology to perform flow simulations in complex geometric configurations. Body-fitted, logically Cartesian meshes are generated to conform to solid surfaces. These grids are allowed to overlap in an arbitrary fashion.

Table 2: OVERFLOW - Average time per step (secs)

No. of Processes	C24	RTJones		Pleiades	
		8 GB/node		*8 GB/node*	
		8ppn	*4ppn*	*8ppn*	*4ppn*
16	6.87	27.71	11.55	16.24	7.29
32	3.74	11.79	5.19	6.96	3.40
64	1.93	5.36	2.96	3.09	1.75
128	1.01	2.37	1.31	1.49	0.91
256	0.51	1.20	0.75	0.74	0.47

OVERFLOW uses finite differences in space and implicit time stepping, with a variety of options for the spatial differencing and for the approximate solution of the large systems that result from implicit time stepping.

The code originated in the heyday of vector machines and the coding style of striving for vectorization has been generally maintained as the code has evolved. Many subroutines of the code operate on arrays in a memory-to-memory fashion, where some small unit of computation is applied to all elements of an array. Profiling of OVERFLOW often shows a small ratio of floating-point operations to memory accesses, and the code tends to be limited by cache size and processor-to-memory bandwidth, not by CPU clock speed.

The test case used in this study, "DLRF6", is a large wing/body/nacelle/pylon test case with 23 zones and 36 million grid points. Table 2 presents the average time per step in seconds (so lower is better) across the three architectures. In all cases the MPI library environment variables MPI_BUFS_PER_HOST and MPI_BUFS_PER_PROC could be set to large enough values so that the results are not affected by retries in the MPI library.

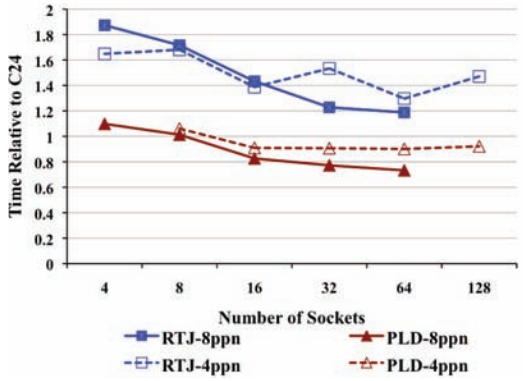


Figure 3: Relative performance of OVERFLOW

The Altix system, C24, has more cache per core, followed by Pleiades and RTJones, so for fully populated runs (8ppn on Pleiades and RTJones), C24 performs best, followed by Pleiades and RTJones. Superlinear convergence can be seen for RTJones and Pleiades, due to increased total cache size and less contention for memory bandwidth. For the 4ppn runs on Pleiades and RTJones performance improves markedly (sometimes by more than a factor of 2) as opposed to 8ppn, due to cache and bandwidth effects. Figure 3 shows that on a socket-per-socket comparison, Pleiades outperforms C24 except for the 4- and 8-socket cases (performance of C24 with 4 sockets, i.e. 8 cores, was 14.88 seconds/step). The performance on Pleiades is better than on RTJones due to larger caches and greater bandwidth to memory on Pleiades.

5. USM3D

USM3D is a 3D unstructured tetrahedral based, cell-centered, finite-volume Euler and Navier-Stokes flow solver [4]. Spatial discretization is accomplished using an analytical reconstruction process for computing solution gradients within tetrahedral cells. Steady and unsteady solutions are supported. The solution is advanced in time to a steady-state condition by an implicit Euler time-stepping scheme. A single-block, tetrahedral, unstructured grid is partitioned into a user-specified number of contiguous partitions, each containing nearly the same number of grid cells. Communication among partitions is accomplished by suitably embedded MPI calls.

The USM3D code is typically memory bound within a single CPU. Available system cache sizes are not adequate to contain a large fraction of the core computation within the cache in the time consuming iterative solve phase. Thus, cache reuse is not significant, and the code is heavily dependent on available memory bandwidth for overall performance. Timing studies show that communication costs between partitions distributed across the nodes are almost negligible. Thus, scaling to large core counts is not related to MPI performance on the system, but is almost always dominated by ever increasing load imbalances created by the partitioning of the grid as the local cell counts decrease and their variance gets larger.

The test case used here is an excerpt from a real production problem computing a high fidelity simulation of a transport aircraft configured for landing. The grid contains 108 million tetrahedra, requiring about 256 GB of memory (2 GB/core) and around 10 GB of scratch online storage.

Table 3 and Figure 4 show the performance of USM3D on the Altix C24 and also on Pleiades and RTJones while running at two different core densities (4ppn and 2ppn). The performance of the code on Pleiades at either core density is significantly faster than C24 as is the 2ppn result on RTJones. At 2ppn the Pleiades system is nearly 60% faster than C24. Even in the worst case, the 4ppn RTJones result is about 90% of the C24 result.

Table 3: USM3D - Average time per step (secs)

No. of Processes	C24	RTJones		Pleiades	
		8 GB/node		*8 GB/node*	
		4ppn	*2ppn*	*4ppn*	*2ppn*
128	9.54	10.4	7.78		6.13
192	6.42	7.06	5.31	5.53	4.14
256	4.93	5.31	4.02	4.18	3.14
384	3.5	3.6	2.72	2.8	2.09
512	2.61	2.79	2.07	2.19	1.65

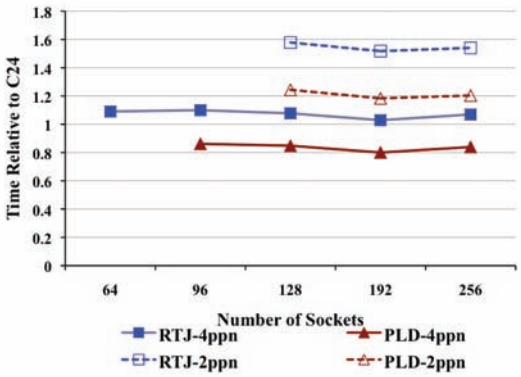


Figure 4: Relative performance of USM3D

Figure 4 shows a socket-based comparison of the performance. The relative performance between runs stays almost flat across widely varying socket (core) counts, demonstrating the code's algorithmic insensitivity to cache size and communication issues. Also, the relative performance for a given socket count with varying core densities is also nearly constant. This is a good sign for the robustness of the architectural design, as the workload and memory access patterns change dramatically as the problem is spread out over more sockets and fewer active cores (implying vastly different styles of sharing of resources).

As can be seen in Table 3, the parallel scaling on all platforms is approximately linear, about 3.6-3.7x from 128 to 512 CPUs. Optimization of the MPI communications has made the code fairly insensitive to the system interconnect performance. This is corroborated by the fact that the Altix 4700 has much better MPI latency performance, yet scaling is essentially invariant across all of the systems.

It is also worth noting that the performance win for USM3D in going from 4ppn to 2ppn on Pleiades is less pronounced than that seen earlier with either CART3D or OVERLFOW when going from 8ppn to 4ppn. The reason is that both OVERFLOW and CART3D gain extensive benefit by effectively doubling their cache size, not sharing cache, and suffering less sharing of memory bandwidth when going from 8pn to 4ppn. In the case of USM3D, the code gains no advantage in cache behavior in going from 4ppn to 2ppn, but does get a slight boost from less sharing of memory bandwidth.

6. CONCLUSIONS

In this paper we presented performance results for three CFD codes on three large compute servers found at the NAS supercomputing facility. The focus of the work was to compare parallel scaling and time to solution between a large shared-memory Altix 4700 system and two clusters of quad-core Xeons connected with InfiniBand. The applications chosen—CART3D, OVERFLOW, and USM3D—represent a large portion of the production use of the NAS facility, and like many other CFD codes, they stress cache and memory bandwidth.

The results show that when compared on a core-per-core basis, the shared-memory Altix usually outperformed the two clusters. Further, we found that contention for the shared resources in the memory hierarchy of the quad-core systems has significant impact on performance. Specifically, reducing contention by leaving cores idle on the quad-core systems lowers the time to solution, but at the expense of requiring twice as many processors. When compared on a socket-per-socket basis, runs on the newer Xeon-based system that leave half the cores idle (4ppn), were on par with the Altix 4700, and were often better. We believe such comparisons are fair because they help compensate for the large price difference between the two types of systems and thus give us a measure of cost-effectiveness.

The approach of leaving cores idle to reduce contention could help improve performance for other CFD codes that stress cache and memory bandwidth. It is worth experimenting with number of processes per node and amount of memory allocated to the MPI library in order to find a cost effective execution configuration.

REFERENCES

1. NAS System Resources: http://www.nas.nasa.gov/Resources/Systems/systems.html
2. Mavriplis, D. J., Aftosmis, M. J., and Berger, M. (2005). High resolution aerospace applications using the NASA Columbia supercomputer, Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, Seattle, WA.
3. Nichols, R. H., Tramel, R. W., and Buning, P. G. (2006). Solver and turbulence model upgrades to OVERFLOW 2 for unsteady and high-speed applications, AIAA 24th Applied Aerodynamics Conference, San Francisco, CA, AIAA-2006-2824.
4. USM3D: http://aaac.larc.nasa.gov/tsab/USM3D/USM3D_52_man.html.

General Performance Optimizations for Several Unstructured Mesh CFD codes on NASA HPC Systems

James Taft
Sienna Software Inc., NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center, Moffett Field, CA 94035, USA

Abstract: Over the past several years there has been a concentrated effort at NASA's Advanced Supercomputing Division (NAS) to dramatically improve the performance of a series of unstructured mesh-based CFD production codes that consume significant fractions of the available cycles on the NAS resident supercomputers . The NAS effort attempts to obtain a "force multiplier" effect through application software upgrades that effectively increase the overall throughput of the NAS systems. while reducing the turnaround time for the engineers utilizing these codes. The results of this effort have shown that there are a number of general platform independent optimizations that are rarely used, but offer significant wins in performance. This paper discusses some of the more successful strategies on significant unstructured mesh NASA CFD codes, USM3D, US3D, and FUN3D. The work described in this paper was performed under sub-contract to CSC Corporation (NASA prime contract number NNA07CA29C).

Keywords: unstructured mesh codes, FUN3D, USM3D, US3D, NAS Pleiades

1. INTRODUCTION

Unstructured mesh CFD codes have enjoyed a recent rapid rise in popularity at NASA's NAS Supercomputing Facility located at the Ames Research Center, Moffett Field, CA. This has been due to the introduction of a number of optimizations that have significantly reduced their memory requirements, and improved their computational efficiency. I/O issues have also been resolved for very large problems. Over the past two years three codes -- USM3D [1], FUN3D [2], and US3D [3] have seen memory reductions of up to 3x and runtime performance improvements of up to 5x. These changes have made unstructured computations affordable, and sometimes as computationally efficient as previous structured code executions. More importantly, the memory reduction changes have allowed users to grow problem sizes significantly. This allows design evaluations that were never possible before. This paper discusses the strategies that were employed in obtaining the improved results. Many of these changes are applicable across a wide range of platforms, and form the basis for an optimization strategy for a more general class of CFD codes.

During the course of our investigation we found a number of impediments to unstructured code efficiency. Several were holdovers from embedded Cray vector programming constructs. Others were poor algorithmic strategies now emphasized by the relatively weak memory systems found in today's clustered micro-processor based architectures. In this paper, we first provide background by way of pointing out differences in structured and unstructured mesh codes and optimization issues arising in modern HPC systems. We then provide details of the optimizations that we performed on USM3D, followed by a brief overview of how these optimizations affect FUN3D and US3D.

2. DIFFERENCES BETWEEN STRUCTURED AND UNSTRUCTURED CODES

There are a number of significant architectural differences between classic structured mesh codes such as OVERFLOW [4], and the typical unstructured mesh codes such as USM3D, FUN3D, and US3D. These differences have been the fundamental inhibitor to the widespread adoption of unstructured codes for years.

The most significant difference has been the much larger memory requirement of unstructured codes. These codes typically store very large block structured coefficient arrays that dominate the memory requirements overall. Where a structured code may need perhaps 20 times the total cell count for storage, the typical unstructured mesh may need over 200 times the cell count. This issue alone prevented adoption of unstructured codes for many years, as problems of interest would simply not fit into the physical memory on the systems.

A second major issue is found in the nature of the memory reference patterns of these codes. Structured mesh codes tend to access data sequentially. They often do this using unit stride approaches, which make better use of cached data and thus achieve a larger fraction of the peak performance of the systems they run on. Unstructured codes, on the other hand, are more random in their memory access patterns. This is the result of their arbitrary cell interconnectivity. As a result, they tend to move through data in the key compute regions with an arbitrary stride. This action guarantees a very poor hit rate in the cache, and less code performance overall. This often makes these codes "unaffordable" relative to the structured mesh equivalents. NAS efforts have dramatically altered this balance in memory and compute for unstructured codes.

3. MODERN HPC SYSTEMS – IMPACT ON OPTIMIZATION

The modern HPC system is typically a cluster of compute nodes incorporating either Intel Xeon or AMD Opteron processors connected via a dedicated InfiniBand (IB) communications network. Users decompose their CFD problems into a number of load balanced spatial sub-domains that execute independently on each processor of the system. These sub-domains occasionally exchange boundary information to synchronize the solution steps using MPI protocols [5] over the IB interconnect.

NASA's main compute cluster, Pleiades, is comprised of 6,400 compute nodes, connected via an InfiniBand based hypercube interconnect. Each node is configured as an 8 core SMP with 8 GB of uniformly accessible ("flat") main memory. A total of 5888 nodes are based on Intel Xeon X5472 3.0GHz0 GHz quad-core processors, while a subset of 512 nodes, called RTJones, are based on Intel Xeon X5355 2.66 GHz quad-core processors.

There are three major issues with the modern clustered systems that significantly impact achievable sustained performance. First, the memory bandwidth in current systems is not sufficient to keep the CPUs busy. Second, the nodal interconnect fabric latency characteristics can inhibit code scaling and overall performance. Third, the I/O sub-system architecture can impact the maximum capabilities of the CFD code due to poor use of buffer space.

4. USM3D

USM3D is a 3D unstructured tetrahedral based, cell-centered, finite-volume Euler and Navier-Stokes flow solver. The USM3D code has been investigated extensively under this optimization effort. There were a number of major issues associated with the code. First, it could not run large problem sizes (>50M cells) due to the large user- defined I/O buffers required in the initialization stage. Second, the code ran very slowly relative to the competing structured mesh codes. A factor of 10x was not uncommon, making it essentially unaffordable for large-scale studies of aero-vehicles at high fidelity.

Memory Reduction – Removal of I/O Buffers Used in Initialization: While USM3D is similar in nature in the flow solver portion of the code to the FUN3D and US3D codes, it does have a unique issue with initialization. Surprisingly, the flow solver was not the memory dominant portion of the code. Code initialization dominated the total memory requirement of the model. In fact, this brief initial peak in memory use during initialization greatly limited the maximum problem size that could be run with the code.

The memory problem was actually the most difficult to address in the USM3D optimization effort, as the reading of the grid/connectivity data and its partitioning across the CPUs involved many thousands of lines of code. User-defined buffers were used to hold transitory data in the master process that was then broadcast to additional large buffers in the remaining processes. The optimization effort focused mainly on reducing the number and size of scratch arrays in the master process and eliminating virtually all of the multiple copies of the data on the outlying processes. by sending only what was needed, rather than globally broadcasting the data, and culling later. In this rewrite, the master was forced to do the culling work for all other processes. Even so, this did not significantly increase the run time in the initialization step.

*Introduction of Integer*2 and Real*4 into the Code:* In the late sixties, HPC users became comfortable with the fact that almost all HPC sites provided 60-bit single precision floating point arithmetic in their CDC supercomputer. Later this precision migrated up to a 64-bit standard for the Cray Research line of supercomputers. These two companies dominated HPC for nearly 30 years. As a result, users became comfortable with this standard of precision. The question of what variables really needed 64-bit precision in a code was left largely unanswered.

Past experience with many CFD flow codes on non-vector architectures has shown that judicious use of Real*4 wherever possible could significantly reduce the runtime and memory requirements of the code. Care must be taken in this process, as it is easy to lose precision in critical areas that will render many solutions useless and codes unreliable. Investigations have shown that much of the work in the USM3D coefficient build and computation of the tendency terms can be stored in Real*4 precision. This implies an immediate reduction by a factor of two in some of the largest arrays in the code. For USM3D, this translated into nearly a factor of two in overall memory requirements as well.

Not only is the use of Real*4 an advantage in memory storage, but it is also offers a serious improvement in effective memory bandwidth (words/second to cache), and effective cache size (twice as many words are stored in cache). Thus, any move to Real*4 can make a significant impact on code performance. For USM3D, the effects of this optimization resulted in run time reductions of around a factor of 2.

For USM3D a number of indices are stored for later use in the indirect addressing of data. Some of these could be converted to Integer*2 to further reduce the memory footprint and improving the cache hit rates on this data as well.

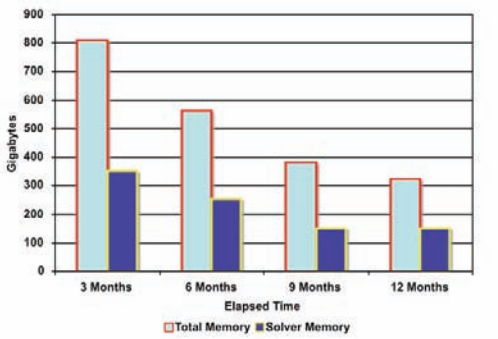


Figure 1: Memory Reduction Time Line for USM3D

Figure 1 depicts the history of the memory reduction effort that was carried out in parallel with the performance enhancement effort. Figure 1 shows that over the course of about one year, the code was able to reduce memory requirements by about a factor of 2 and could in the end run production problems 108M cells in size on systems equipped with memory complements of 2GB/core. The maximum problem size supported was almost 200M cells.

Reduction of Indirect Addressing: The performance enhancement effort began with an attempt to speed up the coefficient build and solver portions of the flow code. Together, these sections consumed about 90% of the code execution time. Since USM3D is an unstructured model, it contained a significant amount of indirect addressing of memory in the coefficient build and point-wise SOR solve. It was found that a reordering of loops in the solver, and restructuring of the storage of intermediate values into unit stride vectors, or even scalars, provided a significant reduction in run-time.

Removal of the Colorization Scheme: The USM3D code has a long heritage. During much of its life it was resident on Cray Research vector machines. During the course of this time, a "colorization" scheme was introduced into the point-wise SOR solve section of the code. This method allows one to vectorize code that could not normally be vectorized. This was a significant win on the vector machines of the past. Another benefit of colorizing the solution algorithm was that it had the potential for accelerating the convergence by increasing the stride between the updated values. In effect, it accelerated the propagation of signals throughout a computational domain.

It was realized early on that this scheme needed to be removed for best performance. This vectorization approach was not needed on non-vector COTS microprocessors, and in fact guaranteed poor performance due to its inherently very low cache reuse. Furthermore, tests revealed that there was no actual benefit in convergence acceleration, or solution stability for a large selection of test cases. A rewrite of the point-wise solve routine NEWRHS to remove this scheme, provided approximately a 2x boost in code performance.

Compilers and Runtime Libraries: Something not mentioned often in optimization efforts is the effect that compiler features and backend code generator limitations can have on overall code performance. We have noted that the compiler is very sensitive to compile time knowledge of loop features, such as loop limits, strides, etc. Loops with variables as starting and stopping points that are unknown at compile time can significantly degrade performance at run time. Loops defined as parameters known at compile time are much more likely to be unrolled effectively and pipelined accordingly. While not a significant factor for USM3D, run time reductions within US3D approached 20% from this restructuring alone.

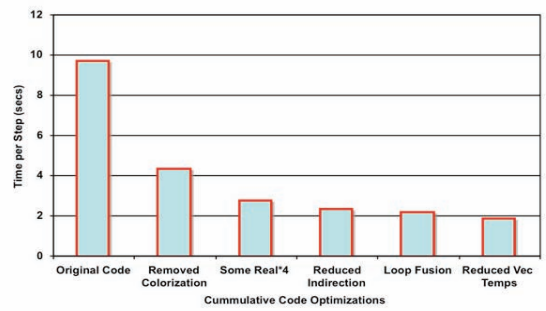


Figure 2: Cummulative Performance of USM3D versus Optimizations (108M cells, 256 CPUs)

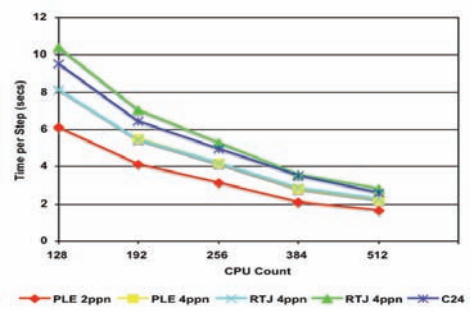


Figure 3: USM3D Performance versus CPU count for Various Architectures (108M cells)

MPI and Scaling: The communication in USM3D was initially highly serialized. It also was organized using Real*8 data transfers. As the nodal data was converted to Real*4, so were the messages to be exchanged. As a result, communication message lengths were reduced by a factor of two. Finally, a more classic scalable peer-to-peer communication scheme was adopted, and MPI scaling became linear, even for the much faster optimized code.

One issue with overall code scalability came to light with the USM3D code. It was noted that code scaling broke down at the higher CPU counts, due to modest variations in the partition sizes found on different CPUs. This pointed out that one must monitor the efficiency of the partitioning code to insure that the load balance remains even as the CPU count increases. It is expected that all unstructured codes will suffer this ultimate limit on scaling for a given problem as partitioning constraints, such as unique boundary conditions, airframe structural considerations, and the like, do not allow complete freedom in how a partition size is determined.

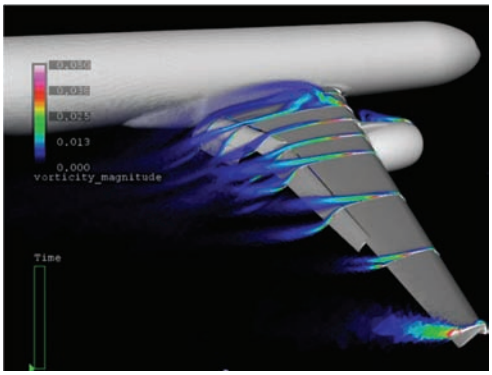


Figure 4 – Notional Aircraft used in the USM3D Optimization Study (108M cells)

Code Status: Figure 2 shows the impact of the major optimizations performed on the USM3D while Figure 3 shows the final performance numbers for various processor-per-node (ppn) counts and across various architectures. Prior to the optimization efforts, this run was impossible. since there simply wasn't enough memory on a node to run it. Note that this run is a real production problem of 108M cells representing a notional transport aircraft. Figure 4 shows a snapshot of the aircraft and selected cuts of the vorticity field. As can be seen in the performance chart, the 2ppn Pleiades runs are significantly faster than the all others, and particularly better than the previous generation Altix shared memory system, C24. The end result of the effort yielded a code that was almost 5x faster with a 2x reduction in overall memory.

5. FUN3D

FUN3D is a next generation unstructured 3D flow solver that can provide many solutions for many flow regimes. It has similar issues with the solve portion that were found in USM3D. Its initialization step however is quite different. FUN3D utilizes an independent pre-processing step to break the initialization data into a series of separate files such that each process reads files that only pertain to its initialization needs. This procedure avoids many of the pitfalls of using a global approach like USM3D in that buffer space for the global array data is non-existent. US3D as noted below, uses the same approach (it should be mentioned that USM3D is moving to this format as well).

FUN3D does however suffer from the same issues in the solve portion of the code. Both on and off diagonal elements were stored at Real*8 precision, as were a number of temporary variables. As with USM3D, the tendency terms were also Real*8 quantities. With the conversion to Real*4 for the off diagonal and tendency terms the solve was sped up by a factor of over 4.5x. The overall code performance was 3.5x faster.

Part of the speedup in FUN3D as in USM3D, was due to efforts to reduce the number indirect addressing activities within the core loops of the solve process. The removal of just one indirect reference can be significant. About half of the time reduction for FUN3D was in the restructuring of key loops to remove such extra references.

6. US3D

US3D is a next generation unstructured CFD code from the University of Minnesota that supports both tetrahedra and rectilinear cells to enhance the resolving power of the simulation for complex geometries. Because of its piecewise structured grid, the code can make use of some line implicit SOR techniques to improve solution convergence rates. Claims are made of 10x speed advantages over point SOR based approaches typically used in codes like USM3D.

The US3D optimization effort has been ongoing for just a few weeks. The solver and viscous flux calculation are the major time consuming components in this code. As with the other codes, the matrix solver portion dominates the run time. Early work identified a number of operations in the solve that could be converted to Real*4. The typical off-diagonal terms were converted first, as were a number of temporary smaller arrays. The end result of this effort to date is 2.6x speedup in the solver and a 2.1x speedup in the code overall for a standard real world test case of approximately 30 million cells. Additional work in the reordering of the indirect addressing approach, should yield at least another factor of two for this code.

7. CONCLUSIONS

Three unstructured mesh CFD codes used by NASA were examined and optimized for execution on NASA's primary HPC supercomputing system. Results were significant, with code capabilities improving significantly. For USM3D, the maximum problem size increased from 49M cells to approximately 200M cells. Runtime performance for the code improved by almost a factor of 5. FUN3D achieved a 3.5x runtime performance improvement with approximately a 30% reduction in its overall memory requirement. The recently begun US3D effort has already achieved a 2.1x speedup and a nearly 30% reduction in overall memory requirements.

The optimizations performed in this effort were generic in nature and are applicable across all currently available COTS based clustered HPC platforms. It was found that the exploitation of Real*4 precision in these codes is a powerful optimization strategy that can allow users to achieve significant speedups in legacy code with little disruption in code structure, and negligible perturbations in solution accuracy. Other changes, such as reduction of indirect addressing in loops, loop fusion, etc can be accomplished quickly as well, with attendant significant runtime reductions.

The current strategy offers significant benefits to the HPC site as well. The force multiplier effects of optimizing large consumers of cycles at these sites, both extends their useful life, and increases their overall throughput. For a particular user, it can enable computations that were heretofore computationally impossible.

In the full paper we will discuss the optimization issues in fuller detail for each code. We will also provide charts of the impact of several of the optimizations in isolation for several of the codes.

REFERENCES

1. USM3D: http://aaac.larc.nasa.gov/tsab/USM3D/USM3D_52_man.html
2. FUN3D: http://fun3d.larc.nasa.gov/
3. US3D: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20070023369_2007022443.pdf
4. OVERFLOW: R.H. Nichols, R.W. Tramel, and P.G. Buning, Solver and Turbulence Model Upgrades to OVERFLOW 2 for Unsteady and High-Speed Applications, AIAA-2006-2824, 24th Applied Aerodynamics Conference, June 2006.
5. MPI: http://en.wikipedia.org/wiki/Message_Passing_Interface

CFD ON GPUS

A Fast Double Precision CFD Code using CUDA

Jonathan M. Cohen *, M. Jeroen Molemaker**

*NVIDIA Corporation, Santa Clara, CA 95050, USA
(e-mail: jocohen@nvidia.com)
**IGPP UCLA, Los Angeles, CA 90095, USA
(e-mail: nmolem@atmos.ucla.edu)

Abstract: We describe a second-order double precision finite volume Boussinesq code implemented using the CUDA platform. We perform detailed validation of the code on a variety of Rayleigh-Benard convection problems and show second order convergence. We obtain matching results with a Fortran code running on a high-end eight-core CPU. The CUDA-accelerated code achieves approximately an eight-time speedup for versus the Fortran code on identical problems. As a result, we are able to run a simulation with a grid of size $384^2 \times 192$ at 1.6 seconds per time step on a machine with a single GPU.

Keywords: CUDA, GPU Computing, Multicore, Rayleigh-Bénard convection.

1. INTRODUCTION

We investigate the use of massively multicore GPUs to accelerate a globally second-order accurate double precision CFD code. While supercomputing clusters allow for large problems to be solved in parallel, the trend towards increased parallelism on a single chip allows researchers to solve larger problems on a single machine than has previously been possible. Our code is implemented using the CUDA platform [1] and is designed to run on the NVIDIA GT200 architecture for GPU computing [3]. The NVIDIA Quadro FX5800 card consists of a single GT200 GPU with 240 cores and 4GB of memory. GT200 supports IEEE-compliant double precision math with peak throughput of 87 GFLOPS/sec. Our code is optimized to take advantage of the parallel GT200 architecture and is approximately 8 times faster than a comparable multithreaded code running on an 8-core dual-socket Intel Xeon E5420 at 2.5GHz. See Table 2 for a summary of relative performance.

2. NUMERICAL METHOD

We solve the incompressible Navier-Stokes equations using the Boussinesq approximation:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla^2\mathbf{u} - \nabla p + \alpha g T \mathbf{z}$$
$$\frac{\partial T}{\partial t} = -(\mathbf{u}\cdot\nabla)T + \kappa\nabla^2 T$$
$$\nabla\cdot\mathbf{u} = 0$$

where $\mathbf{u} = (u, v, w)$ is the fluid velocity field, T is the fluid temperature, p is the fluid pressure field, α is the coefficient of thermal expansion, g is the magnitude of gravity, ν is kinematic viscosity, and κ is thermal diffusivity.

We solve these equations on a staggered regular grid (Arakawa C-grid) using a second order finite volume discretization. The advection terms are discretized using centered differencing of the flux values, resulting in a discretely conservative second-order advection scheme. All other spatial terms are discretized with second-order centered differencing. We use a second-order Adams-Bashford method which we prime with a single forward-Euler step at the start of the numerical integration. Pressure is treated via a projection method, where pressure is calculated instantaneously at the end of the time step to enforce $\nabla\cdot\mathbf{u} = 0$. This requires solving a Poisson equation for pressure, for which we use a multigrid method.

While we have chosen simple discretizations for ease of implementation and validation, our code is designed to support a wide variety of higher-order discretizations and stencils. Our optimization strategies will apply to more complex higher order numerical methods as well.

3. IMPLEMENTATION

We have explored a number of GPU-specific optimization strategies. GT200 is designed to run tens of thousands of threads simultaneously, using the large amount of parallelism to hide latencies for off-chip memory reads and writes. GT200 has peak off-chip memory bandwidth of approximately 102 GB/sec, which corresponds to an optimal balance of just over 6 math operations per double precision value loaded from memory. Because most of our inner loops perform a small amount of math, performance of our code is mainly limited by memory bandwidth. Therefore we focused on optimizing our memory access patterns to take advantage of GT200's streaming memory architecture.

3.1 Access Pattern Optimization

On GT200, threads are grouped into batches of 32 called *warps* that execute in lockstep SIMD fashion. If threads in a warp read from the same cache line in the same cycle, these reads are batched into a single operation via a process known as *memory coalescing*. Coalescing operates at half-warp granularity, so uncoalesced loads and stores waste $15/16^{\text{ths}}$ of available memory bandwidth. Therefore the most important optimization for memory-bound applications is to arrange work so that threads in the same warp will access sequential memory locations at the same time.

GT200 has two small on-chip caches: a read-only L1 cache called the *texture cache*, and a read/write software managed cache called *shared memory*. With thousand of simultaneous active threads, on-chip caches are beneficial only if threads scheduled to the same processor access the same cache lines at the same time. Therefore optimizing for cache performance is very similar to optimizing for memory coalescing. The texture cache can be thought of as a "bandwidth aggregator" because it is designed to aggregate memory requests over several cycles so that coalescing will be more efficient. For some routines, especially finite difference stencils, we found the texture cache to yield a performance improvement of up to 1.4x. In many cases, however, it produced no benefit over coalescing alone. Because the shared memory cache is software managed, we found that the cost of the logic required to use it effectively typically was not offset by the bandwidth savings.

3.2 Congruent Padding

Another optimization we found to be important was a concept we term "congruent padding." By congruent, we mean that for all pairs of indices (i, j, k), the offset in bytes between the memory location of element (0, 0, 0) and element (i, j, k) should be the same for all grids. Figure 1 demonstrates congruent padding in two dimensions. Because GT200 runs most efficiently with a large numbers of threads, we assign one thread per computational cell (i, j, k). Since all threads translate from grid indices to memory locations in parallel, index translations must be recalculated at every grid cell. Congruent padding amortizes the cost of index translation over several grids by calculating the offset from location (0, 0, 0) to (i, j, k) once per thread, and then adding this offset to the base pointer for each grid. In our code, this optimization reduces the number of instructions of a typical GPU routine by 10-15% and reduces the register usage by 0-10%. Minimizing the number of per-thread registers allows for more threads to be active at once. Having more active threads covers memory latencies more effectively, which improves throughput [3].

(-1,2) 18	**(0,2)** 19	**(1,2)** 20	**(2,2)** 21	22	23
(-1,1) 12	**(0,1)** 13	**(1,1)** 14	**(2,1)** 15	16	17
(-1,0) 6	**(0,0)** 7	**(1,0)** 8	**(2,0)** 9	10	11
(-1,-1) 0	**(0,-1)** 1	**(1,-1)** 2	**(2,-1)** 3	4	5

(a) A 2x2 grid with 1 row of ghost cells on all sides.

18	**(0,2)** 19	**(1,2)** 20	**(2,2)** 21	**(3,2)** 22	23
12	**(0,1)** 13	**(1,1)** 14	**(2,1)** 15	**(3,1)** 16	17
6	**(0,0)** 7	**(1,0)** 8	**(2,0)** 9	**(3,0)** 10	11
0	1	2	3	4	5

(b) A 4x3 grid with no ghost cells.

Fig 1: An example of congruent padding in 2 dimensions. All grids have the same physical layout in memory, even though they may have different logical dimensions. Computational cells are white, ghost cells are light gray, and unused padding is dark gray.

Resolution	Full Slip		No Slip	
	Value	Difference	Value	Difference
$16 \times 8 \times 16$	659.69	-	1674.29	-
$32 \times 16 \times 32$	658.05	1.64	1699.25	24.96
$64 \times 32 \times 64$	657.65	0.40	1705.59	6.34
$128 \times 64 \times 128$	657.54	0.11	1707.22	1.63
∞	657.51	-	1707.76	-

Table 1: Calculated critical Rayleigh values for full-slip (aspect ratio $\sqrt{2} : .5 : 1$) and no-slip (aspect ratio $\pi : .5 : 3.11$) boundaries at different resolutions. The columns labeled Difference show the differences in values between subsequent grid resolutions. The reduction of this error by a factor of 4 for each doubling of resolution shows the globally second order convergence character of the numerical discretizations. The last row shows the Richardson extrapolated values, which match theory.

4. VALIDATION

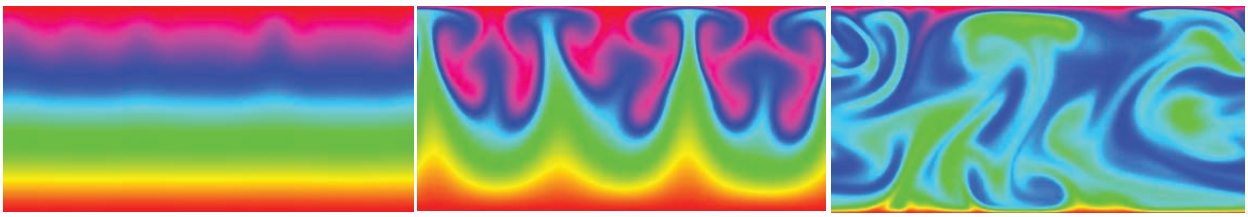
To demonstrate the potential of our GPU based code for scientific applications we have validated the code for a range of problems. We compared our results with an existing CPU based code written in Fortran [5] as well as against published analytical, numerical, and experimental results. Since our code implements the Boussinesq equations we choose to examine whether it can reproduce known solutions to different Rayleigh-Bénard convection problems in which a constant temperature difference ΔT is maintained between the top and bottom boundaries of the domain. The most basic result for Rayleigh-Bénard convection is the critical value of the dimensionless Rayleigh number $Ra = g\alpha\Delta T/\kappa\nu$. Below the critical value Ra_c the solution is motionless and heat flux between top and bottom is purely diffusive. When $Ra > Ra_c$ the diffusive solution becomes unstable to perturbations of arbitrarily small amplitude and a solution with non-trivial flow and enhanced vertical heat transport ensues.

We estimated Ra_c in our codes by calculating positive and negative growth rates of **u** for small perturbations around Ra_c and extrapolating to find the value of Ra for which the growth rate would be zero. Our GPU and CPU codes use identical numerical methods and therefore have matching values to several decimal places. Table 1 shows calculated Ra_c values from our codes. The third and fifth columns show the differences in Ra_c obtained for subsequent resolutions, highlighting the globally second order convergence rate of the numerical implementation. For the 2D problem with the aspect ratios chosen, analytical values are known [7] (we treat this as a 3D problem by choosing a

smaller aspect ratio and periodic boundaries in the y dimension). Using Richardson extrapolation, we obtain a value of Ra_c = 657.51 for the full-slip case, and Ra_c = 1707.76 for the no-slip case, both of which match the analytical results.

To test a fully 3D problem, we also studied the onset of convection in a cubic box with no-slip conditions on all sides and Dirichlet conditions for T on the side boundaries. We find a critical Rayleigh number Ra_c = 6755 for this case, matching the published experimental [2] and numerical [4, 6] values. To verify the nonlinear advection terms in the equations, we calculated the solution for a supercritical value of $Ra = 4.4 \times 10^4$. We then calculated the Nusselt number $Nu = \overline{(wT + \kappa T_z)}/(\kappa \Delta T)$. The Nusselt number is the ratio of vertical heat transport to diffusive heat transport across a 2D interface of a motionless solution. Nu also depends on the Prandtl number, $Pr = \nu/\kappa$. For $Ra = 4.4 \times 10^4$ and $Pr = 0.71$, Nu computed at the upper and lower boundaries is 2.05 (using both CPU and GPU codes) and exhibits global second order convergence when computed at increasing resolutions. This matches published results [6].

5. COMPARATIVE PERFORMANCE



(a) Perturbations appear in the stratifications.

(b) Instabilities form.

(c) Temperature mixes after the onset of turbulence.

Fig 2: False color plot of T at the $y = 0$ plane for a $384^2 \times 192$ resolution simulation with $Ra = 10^7$.

To generate a timing comparison, we ran an unsteady Rayleigh-Bénard convection problem on our GPU and CPU codes with $Ra = 10^7$ and $Pr = .71$. The simulation domain was set to [-1, 1] × [-1, 1] × [-.5, .5], with periodic boundary conditions in x and y, and no-slip boundaries in z. As shown in Figure 2, the flow starts out motionless until instabilities form, and then transitions to turbulence. To accelerate convergence of the multigrid solver for pressure, we reuse the solution from the previous time step as an initial guess. Consequently, the number of v-cycles required for convergence increases as the flow becomes less steady. In order to characterize our performance fairly, we only count the average time per step once the number of v-cycles per step has stabilized. Because of the different performance characteristics of the GPU and the CPU, we have chosen slightly different multigrid relaxation schemes for the two codes. The GPU code, which uses a red-black Gauss-Seidel point relaxer, requires 1 full multigrid step followed by 7 v-cycles. The CPU code uses a red-black line relaxer and requires 1 full multigrid step followed by 13 v-cycles.

Table 2 shows the relative timing of the two codes for problem sizes up to $384^2 \times 192$, which is the largest that can fit on a single GPU with 4GB of memory. GPU times are for a single Quadro FX5800 running on a Core2-Duo E8500 at 3.17GHz. CPU times are for an 8-core dual socket Xeon E5420 at 2.5GHz, and the CPU code is multithreaded using OpenMP and MPI to take advantage of all 8 cores. These hardware configurations were chosen because they are roughly equivalent in terms of cost.

Our GPU timing results are significant for two reasons. First, at the high end of simulation size, $384^2 \times 192$, we have achieved performance on a single node comparable to that often achieved on a small cluster. Second, turbulent features can be observed at resolutions as low as $128^2 \times 64$. At this resolution, our code can calculate up 12 time steps per second, which enables interactive turbulent simulations on a commodity PC with a single GPU.

Resolution	GPU CUDA Code			CPU Fortan Code			**GPU Speedup**
	ms/Step	ms/Step/Node	Scaling	ms/Step	ms/Step/Node	Scaling	
$64^2 \times 32$	24	18.3e-5	-	47	37.0e-5	-	**2.0x**
$128^2 \times 64$	79	7.5e-5	0.41x	327	31.2e-5	0.84x	**5.3x**
$256^2 \times 128$	498	5.9e-5	0.79x	4070	48.5e-5	1.55x	**8.2x**
$384^2 \times 192$	1616	5.7e-5	0.97x	13670	48.3e-5	1.00x	**8.5x**

Table 2: Relative performance for the $Ra = 10^7$ problem. Time per step is calculated ignoring the initial run-up to turbulence since multigrid converges faster during this period. The column labelled Scaling indicates the change in time per step per computational node from one resolution to the next. Linear scaling in the number of computational nodes would therefore be 1.0x.

5. FUTURE WORK

Our results demonstrate that GPU-based codes can be a powerful tool for real scientific applications. We have demonstrated second order convergence in double precision on buoyancy-driven turbulence problems using a GPU. Using a single workstation that is equipped with a GPU, our code can integrate a non-trivial flow at moderately high resolutions up to 8 times faster than a high-end 8-core CPU. We intend to extend our work in several ways. First, we will implement higher-order methods in both space and time. Second, we are interested in numerical ocean and atmospheric models such as [8] that use logically regular grids, but are geometrically irregular. Third, we will explore multiple GPU configurations such as [9]. A single motherboard may have several PCI-express buses, each of which can connect to one or more GPUs. In addition to improving performance for large computing clusters, this has the potential to dramatically increase the resolution that people without access to clusters can achieve.

REFERENCES

1. NVIDIA Corporation (2008). *CUDA programming guide, version 2.0*.
2. Leong, W.H., Hollands, K.G.T., and Brunger, A.P. (1998). On a physically realizable benchmark problem in internal natural convection. *Int. J. Heat Mass Transfer* 41, 3817-3828.
3. Lindholm, E., Nickolls, J., Olberman, S., and Montrym, J. (2008). NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro* 28(2), 39-55.
4. Mizushima, J. And Matsuda, O. (1997). Onset of 3d thermal convection in a cubic cavity. *J. Phys Soc. Japan* 66, 2237-2341.
5. Molemaker, M.J., McWilliams, J.C., and Capet, X. (2009). Balanced and unbalanced routes to dissipation in equilibrated Eady flow. *J. Fluid Mech,* In press.
6. Puigjaner, D., Herrero, J., Simo, C., and Giralt, F. (2008). Bifurcation analysis of steady Rayleigh-Bénard convection in a cubical cavity with conducting sidewalls, *J. Fluid Mech* 598, 393-427.
7. Reid, W.H. and Harris, D.L. (1958). Some further results on the Bénard problem, *Phys. Fluids* 1, 102-110.
8. Shchepetkin, A.F. and McWilliams, J.C. (2005). The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model, *Ocean Modelling* 9, 347-404.
9. Thibault, J.C. and Senocak, I (2009). CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows, *47th AIAA Aerospace Sciences Meeting*, Orlando, Florida, paper no: AIAA 2009-758.

Acceleration of a CFD Code with a GPU

Dennis C. Jespersen

NASA/Ames Research Center Moffett Field, CA 94035, USA
(e-mail:Dennis.Jespersen@nasa.gov)

Abstract: The CFD code OVERFLOW includes as one of its solver options a quasi-SSOR algorithm. This is a fairly small piece of code but it accounts for a significant portion of the total computational time. This paper studies some of the issues in accelerating the code by use of a GPU. The algorithm needs to be modified to be suitable for a GPU, and attention needs to be given to 64-bit and 32-bit arithmetic.
Keywords: GPU, acceleration

1. INTRODUCTION

Computational Fluid Dynamics has a history of seeking and requiring ever higher computational performance. This quest has in the past been satisfied mainly by faster clock speeds. The era of increasing clock rates has ended, due mainly to heat dissipation constraints. A boost in computational performance without increasing clock speed can be supplied by parallelism. This parallelism can come in the form of shared memory parallelism, distributed memory parallelism, or a combination of the two. Common current paradigms for parallelism are explicit message-passing with MPI [12] for either distributed or shared memory systems and OpenMP [13] for shared memory systems. A hybrid paradigm is also possible.

A further possibility for code acceleration has recently attracted much attention. A Graphics Processing Unit (GPU) can offer a very high computational rate if an algorithm is well-suited for the device. There have been several projects illustrating the acceleration of scientific computing codes that is possible by using GPUs [2,6,10]. In this paper we study the issues in accelerating a well-known CFD code, OVERFLOW, on a GPU.

2. OVERFLOW CODE

The OVERFLOW code [3,4,9] is intended for the solution of the Reynolds-averaged Navier-Stokes equations with complex geometries. The code uses finite differences on logically Cartesian meshes. The meshes are body-fitted and geometric complexity is handled by allowing the meshes to arbitrarily overlap one another. The code can be run in "standard" mode by having the user supply all the meshes and the interconnection information, or it can be run in "OVERFLOW-D" [5] mode by having the user supply only near-body meshes along with an "X-ray" file which helps to define the solid surfaces. In "OVERFLOW-D" mode the code itself generates off-body Cartesian meshes as well as interpolation information allowing one mesh to obtain data from another.

OVERFLOW uses implicit time-stepping and can be run in time-accurate or in steady-state modes. Implicit time-stepping is used because implicit methods tend to mitigate severe stability limits on the size of the time step that arise for explicit methods on highly-stretched grids that are common for viscous flow problems at high Reynolds numbers. A consequence of implicit time-stepping is that some method is needed to approximately solve the large system of equations that arises when marching from one time level to the next.

The OVERFLOW user needs to specify physical flow inputs such as Mach number and Reynolds number, and boundary conditions which typically define solid walls and inflow or outflow regions. Along with these physics-type inputs there are inputs which choose particular numerical algorithms and specify parameters for them.

The basic equation of fluid motion solved by OVERFLOW is of the form

$$Q_t + L(Q) = f(Q), \tag{1}$$

where Q are the flow variables, $L(Q)$ denotes all the spatial differencing terms, and $f(Q)$ denotes terms from boundary conditions and possible source terms. As well there is turbulence modeling, but we do not consider that in this paper.

The basic equation (1) is written in "delta form" [1,8]

$$A(\Delta Q^{n+1}) = R^n, \tag{2}$$

where A is a large sparse matrix which is not explicitly constructed, $\Delta Q^{n+1} = Q^{n+1} - Q^n$, and R^n involves the discretization of the $L(Q)$ terms at time level n. The user of OVERFLOW must choose among several possible discretizations (e.g. central differencing, Total Variation Diminishing, Roe upwind). Each of these choices typically requires further user specification of numerical parameters, e.g. dissipation parameters or type of flux limiter and parameters for the limiter. Finally the user needs to decide which implicit algorithm to use: some choices are factored block tridiagonal, factored scalar pentadiagonal, LU-SGS. Over the years the code evolved and expanded to present 6 basic choices for the implicit part of the algorithm.

3. GPU CONSIDERATIONS

GPU cards were originally hard to program and had a steep learning curve. The advent of less daunting interfaces such as CUDA [11] has led to an explosion of interest in using GPUs for numerically intensive work in scientific computing.

For our purposes here the key issues of GPU cards are massive parallelism (hundreds or thousands of threads), strict SIMD parallelism, 32-bit floating point arithmetic, and the overhead of data traffic between the CPU and GPU. For a GPU to successfully accelerate a piece of code the code must be amenable to large-scale SIMD parallelism, must tolerate 32-bit floating-point arithmetic, and must contain enough computational work to amortize the cost of transferring data from the CPU to the GPU and transferring results back from the GPU to the CPU. (Recent GPU hardware supports some limited 64-bit arithmetic but 32-bit arithmetic is significantly faster.) We will see that the SSOR algorithm in OVERFLOW is not well-suited to a GPU, but that a Jacobi version of the algorithm might be suitable for a GPU.

4. THE SSOR ALGORITHM IN OVERFLOW

In an attempt to ease the user's burdensome task of selecting algorithm options and choosing parameters which may change for each class of flow problem, recently another option was added for the implicit part of OVERFLOW with the hope that it would be widely applicable and would be almost universally usable [7]. This algorithm is called in the references an SSOR algorithm, but it is strictly speaking a mix of an SSOR algorithm and a Jacobi algorithm, a "quasi-SSOR" algorithm.

The key step of the quasi-SSOR algorithm is as follows. At each grid point with index (j,k,l) one computes 5×5 matrices AJ, AK, AL, CJ, CK, CL. Then, with iteration stage denoted by a superscript and with a relaxation parameter ω, relaxation steps are of the form

$$\begin{aligned} \Delta Q^{n+1}_{jkl} &= (1-\omega)\Delta Q^n_{jkl} + \omega(R^n_{jkl} - AJ_{jkl}\Delta Q^n_{j-1,k,l} - AK_{jkl}\Delta Q^{n+1}_{j,k-1,l} - AL_{jkl}\Delta Q^{n+1}_{j,k,l-1} \\ &\quad -CJ_{jkl}\Delta Q^n_{j+1,k,l} - CK_{jkl}\Delta Q^n_{j,k+1,l} - CL_{jkl}\Delta Q^n_{j,k,l+1}) \end{aligned} \tag{3}$$

for a forward sweep (assuming the 5-vectors $\Delta Q^{n+1}_{j,k-1,l}$ and $\Delta Q^{n+1}_{j,k,l-1}$ have been computed, and updating all ΔQ_j as soon as a full line of j values has been computed), and a step of the form

$$\begin{aligned} \Delta Q^{n+1}_{jkl} &= (1-\omega)\Delta Q^n_{jkl} + \omega(R^n_{jkl} - AJ_{jkl}\Delta Q^n_{j-1,k,l} - AK_{jkl}\Delta Q^n_{j,k-1,l} - AL_{jkl}\Delta Q^n_{j,k,l-1} \\ &\quad -CJ_{jkl}\Delta Q^n_{j+1,k,l} - CK_{jkl}\Delta Q^{n+1}_{j,k+1,l} - CL_{jkl}\Delta Q^{n+1}_{j,k,l+1}) \end{aligned} \tag{4}$$

for a backward sweep (again assuming $\Delta Q^{n+1}_{j,k+1,l}$ and $\Delta Q^{n+1}_{j,k,l+1}$ have been computed). The forward/backward pair is then iterated. This algorithm is not strictly speaking an SSOR algorithm; it is Jacobi in J-lines and SSOR in K and L planes. We will refer to it as SSOR for simplicity. This algorithm needs the 6 nearest spatial neighbors of ΔQ^n_{jkl} at either iteration level n or $n+1$.

The SSOR algorithm is a modest-sized subroutine but it may consume 80% of the total runtime of the code, so it is a computational hot spot. The modest size of the subroutine and the large fraction of total time consumed by the algorithm make using a GPU as a coprocessor to accelerate the code an attractive idea.

Unfortunately, the algorithm as it stands is not suited to a GPU due to the dependencies of the iteration (ΔQ^{n+1} appearing on the right-hand side of equations (3) and (4)). An algorithm that would be suited to a GPU would be a Jacobi algorithm where relaxation steps are of the form

$$\begin{aligned}\Delta Q^{n+1}_{jkl} &= (1-\omega)\Delta Q^{n}_{jkl} + \omega(R^{n}_{jkl} - AJ_{jkl}\Delta Q^{n}_{j-1,k,l} - AK_{jkl}\Delta Q^{n}_{j,k-1,l} - AL_{jkl}\Delta Q^{n}_{j,k,l-1} \\ &\quad -CJ_{jkl}\Delta Q^{n}_{j+1,k,l} - CK_{jkl}\Delta Q^{n}_{j,k+1,l} - CL_{jkl}\Delta Q^{n}_{j,k,l+1}) \end{aligned} \tag{5}$$

Here we could envision assigning a thread of computation to each grid point and the threads could compute independently of one another because there are no ΔQ^n terms on the right-hand side of (5).

It is important to realize that the Jacobi algorithm might be less robust or might converge slower than the original SSOR algorithm. Fully discussing this would take us too far afield; in this paper our focus is on acceleration. If significant speedup can be attained then it will be important to visit stability and convergence questions for the Jacobi algorithm.

The work presented here proceeded in several stages:

1. Implement a Jacobi algorithm on the CPU using 64-bit arithmetic; compare performance and convergence/stability of Jacobi and SSOR.
2. Implement a Jacobi algorithm on the CPU using 32-bit arithmetic; compare performance and convergence/stability of 64-bit and 32-bit Jacobi.
3. Implement a Jacobi algorithm on the GPU; compare performance of the GPU algorithm with the 32-bit CPU algorithm.

5. IMPLEMENTATION AND RESULTS

The work here was done on a workstation equipped with a quad-core AMD Opteron 2352 processor (2.1GHz, 512KB L2 cache for each core, 2 MB shared L3 cache, 1 GHz HyperTransport). Only one core was utilized for the computation. The host compiler system was the Portland Group compiler suite version 8. The GPU card was an NVIDIA GeForce 8800 GTX with NVIDIA Driver Version 173.14.12. The connection between CPU and GPU was a PCI Express 16X bus. The programming interface was CUDA version 1.0.

Implementing the Jacobi algorithm on the CPU, both in 64-bit and 32-bit arithmetic, was simple. The strategy for GPU implementation of the Jacobi algorithm was to compute all the matrices AJ, etc., on the CPU and transfer them to the GPU. The Jacobi algorithm itself was hand-translated into CUDA code. Each column of data (j and k fixed, l varying) was assigned to one computational thread.

The test case is a viscous laminar flow over a flat plate; this is a three-dimensional extension of a classic fluid dynamics flow problem. The grid was size $41 \times 21 \times 61$. We compare the SSOR algorithm, the Jacobi algorithm in 64-bit arithmetic, the Jacobi algorithm in 32-bit arithmetic, and the Jacobi algorithm with GPU implementation. First we show the residuals for these cases; see Figure 1. At least for this case, 32-bit arithmetic on the left-hand and 64-bit arithmetic on the right-hand side is sufficient to attain full 64-bit accuracy of the solution; the flow solutions for the 4 variants differ from one another at the level of 64-bit arithmetic roundoff error. Also, all the Jacobi residual curves, though different in details, would plot atop one another if they were plotted on the same graph, while the SSOR algorithm converges slightly faster than the Jacobi algorithm.

Here is performance data for these algorithmic variants. We show two grid sizes, $41\times21\times61$ and $121\times41\times81$. Times shown are seconds per step for the whole code, so they include all explicit and implicit operations and all data transfer to and from the GPU. For the $41 \times 21 \times 61$ grid, compute time on the GPU was 0.072 sec/step and total GPU time, including data transfer, was 0.092 sec/step. For the $121\times41\times81$ grid, compute time on the GPU was 0.590 sec/step and total GPU time, including data transfer, was 0.719 sec/step.

Algorithm	$41 \times 21 \times 61$	$121 \times 41 \times 81$
SSOR	0.7020 sec/step	6.2588 sec/step
Jacobi/64	0.7522 sec/step	6.5258 sec/step
Jacobi/32	0.5733 sec/step	4.9104 sec/step
Jacobi+GPU	0.4078 sec/step	3.5561 sec/step

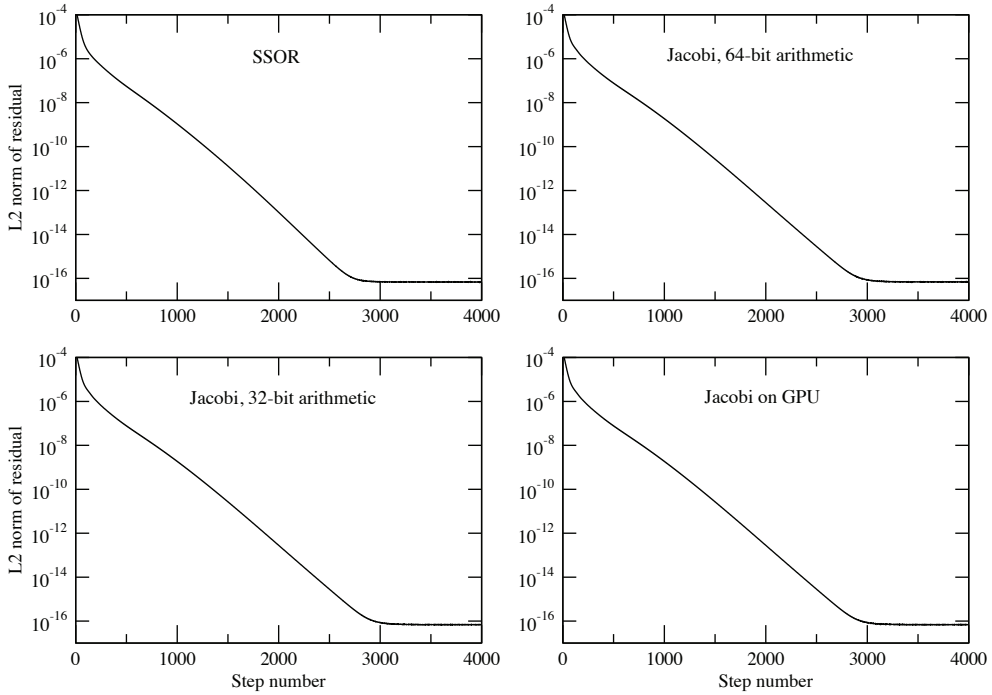


Figure 1: Convergence for laminar flat plate

6. DISCUSSION AND CONCLUSIONS

These first results are encouraging. For a modest effort and for a naive implementation of the Jacobi algorithm on the GPU (attention was paid to correctness, no effort was expended on GPU optimization) an acceleration was attained. It is likely that an optimization effort on the GPU will yield further performance gains. For example, the current implementation on the NVIDIA card is strictly memory-to-memory, with no use of the faster local memory on each multiprocessor. Intelligent use of the local memory should improve the GPU performance.

A wider variety of test cases needs to be tried with the Jacobi algorithm (both 64-bit and 32-bit arithmetic), to see if any surprises arise in terms of stability and convergence. It is possible that that the Jacobi 32-bit algorithm will have a more limited domain of applicability than the 64-bit SSOR algorithm. In any event, there probably are many cases in which the Jacobi 32-bit algorithm with GPU acceleration will be useful in improving the performance of OVERFLOW.

ACKNOWLEDGMENT

This work is partially supported by the Fundamental Aeronautics program of NASA's Aeronautics Research Mission Directorate.

REFERENCES

1. Beam, R., and Warming, R.F. (1976). An implicit finite-difference algorithm for hyperbolic systems in conservation law form. *J. Comp. Physics*, Vol. 22, pp. 87–110.
2. Brandvik, T., and Pullan, G. (2008). Acceleration of a 3D Euler solver using commodity graphics hardware, *AIAA 46th Aerospace Sciences Meeting*, Reno, NV, paper no. AIAA-2008-607.
3. Buning, P. G., Chiu. I. T., Obayashi, S., Rizk, Y. M., and Steger, J. L. (1988). Numerical Simulation of the Integrated Space Shuttle Vehicle in Ascent, *AIAA Atmospheric Flight Mechanics Conference*, paper no. 88-4359-CP.
4. Kandula, M. and Buning, P. G. (1994). Implementation of LU-SGS algorithm and Roe upwinding scheme in OVERFLOW thin-layer Navier-Stokes code, *AIAA 25th Fluid Dynamics Conference*, Colorado Springs, CO, paper no. AIAA-94-2357.

5. Meakin, R. L. (2001). Automatic off-body grid generation for domains of arbitrary size, *AIAA 15th Computational Fluid Dynamics Conference*, Anaheim, CA, paper no. AIAA-2001-2536.
6. Michalakes, J. and Vachharajani, M. (2008). GPU acceleration of numerical weather prediction, *Parallel Processing Letters*, Vol. 18, No.4, pp. 531–548.
7. Nichols, R. H., Tramel, R. W., and Buning, P. G. (2006). Solver and turbulence model upgrades to OVERFLOW 2 for unsteady and high-speed applications, *AIAA 24th Applied Aerodynamics Conference*, San Francisco, CA, paper no. AIAA-2006-2824.
8. Pulliam, T. H. and Chaussee, D. S. (1981). A diagonalized form of an implicit approximate factorization algorithm, *J. Comp. Phys.*, Vol. 39, pp. 347–363.
9. Renze, K. J., Buning, P. G., and Rajagopalan, R. G. (1992). A comparative study of turbulence models for overset grids, *AIAA 30th Aerospace Sciences Meeting*, Reno, NV, paper no. AIAA-92-0437.
10. Thibault, J. C. and Senocak, I. (2009). CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows, *AIAA 47th Aerospace Sciences Meeting*, Orlando, FL, paper no. AIAA-2009-758.
11. `http://www.nvidia.com/object/cuda_home.html`.
12. `http://www-unix.mcs.anl.gov/mpi`.
13. `http://www.openmp.org`.

Application of a Kinetic Theory based solver of the Euler Equations using GPU

MATTHEW R. SMITH* (National Centre for High Performance Computing, Hsinchu, Taiwan.
msmith@nchc.org.tw)

FANG-AN KUO (National Centre for High Performance Computing, Hsinchu, Taiwan.
mathppp@nchc.org.tw)

CHAU-YI CHOU (National Centre for High Performance Computing, Hsinchu, Taiwan.
b00cyc00@nchc.org.tw)

JONG-SHINN WU (National Chiao Tung University, Hsinchu, Taiwan.
chongsin@faculty.nctu.edu.tw)

HADLEY M. CAVE (University of Canterbury, New Zealand
hmcave@canterbury.edu.nz)

Keywords: GPU, Euler solver, Kinetic Theory, Finite Volume Method, CFD.

Overview

Presented is a modified form of the Quiet Direct Simulation (QDS) method [1] adapted for application of Graphics Processing Units (GPU) for flux calculation. Fluxes between source and destination cells calculated by QDS are flux-vector split and (on a regular Cartesian grid) a function of the source cell alone. The resulting advantage is the rapid calculation of fluxes between cells without the prior exchange of information between them, allowing highly efficient calculation using GPU. Various flow problems have been solved and consistent speed-ups of over 35 times (when compared to an equivalent single CPU code) are reported.

Introduction

The use of Graphics Processing Units (GPU's) to assist in the solution of various engineering problems is hardly recent [2,3]. The solution to the Euler Equations is an example of a problem which still maintains relevance in modern engineering problems. This hyperbolic set of partial differential equations possesses analytical solutions in the rarest (and often least useful) conditions and so a significant amount of effort has been spent on their numerical solution. The increase in application of Computational Fluid Dynamics (CFD) over the recent decades has lead to a large number of mathematical and physically based numerical solutions to the Euler Equations [2,3].

Numerous numerical methods have been used in conjunction with GPU technology to solve the Euler equations and have all demonstrated, to varying extent, considerable speed up when compared to existing single CPU codes. Elsen *et.al.* [3] applied a vertex-based finite difference method (with a multi-grid scheme) to the solution of hypersonic flows. However, possibly the most common of these numerical methods is the Finite Volume Method (FVM) [3,4]. In a majority of cases the fluxes are calculated using flux-difference splitting which requires knowledge of conditions on both sides of a cell interface, used together simultaneously, to calculate a net flux across a surface [5]. Such solvers work by calculating the conditions normal to cell interfaces and calculating a series of one- dimensional fluxes across each. This concept is known as direction decoupling and has been shown to result in errors when the flow is not aligned with the computational grid [5].

An alternative are flux-vector split methods, where fluxes are linearly separated into components from both sides of a cell interface and can be calculated separately. These methods are infamous for their typical excessive numerical dissipation. While there are many mathematically split methods available, various kinetic theory based schemes have emerged. Such schemes base fluxes on the mathematical interpretation of phenomenological models. Possibly the most well-known kinetic-theory based finite volume solution method is Pullin's Equilibrium Flux Method [6]. The fluxes of EFM were derived by taking moments of the Maxwell-Boltzmann equilibrium distribution function at the cell interfaces. The resulting flux required the evaluation of two moments – one from each side of the interface – with the net flux across the surface the difference between each. The validity of the method lies in the assumption of flow being divided into a collision phase and a free molecular flight phase. During free m molecular flight, no forces are placed upon fluxing particles as they move from their source region to their destination. These fluxes are calculated based on the conditions normal to the interface; extensions to two dimensional flows were still performed using a series of one- dimensional fluxes.

A more general form of EFM was developed by Smith *et al.* [5] taking into account the true direction nature of the equilibrium fluxes. This method, called TDEFM (True Direction Equilibrium Flux Method) provided the analytical solution to gaseous motion from one cell to an arbitrary destination cell, regardless of whether or not these cells share an adjacent interface. The primary disadvantage to this method was the large computational expense associated with the evaluation of multiple error and exponential functions. As an alternative, the Quiet Direct Simulation (QDS) method has been developed in its current form by Smith *et al.* [1] and is based on the particle-based QDSMC method of Albright *et al.* as a method for simulating plasmas and for Eulerian flow [7,8]. The fluxes obtained by QDS are approximations of the TDEFM fluxes which avoid the evaluation of any expensive mathematical functions while retaining the true direction quality of TDEFM.

Here, the QDS algorithm is slightly modified and applied to employ the Graphics Processing Units (GPU's) of a typical video card using the CUDA library. The nature of QDS makes it ideal for such calculation and results demonstrate significant speedup when compared to a similar (non-GPU) code. Presented here is a description of the QDS algorithm, simulation and code validation using several standard benchmark problems. Finally, details of the computational expense associated with each code are presented.

Quiet Direct Simulation Algorithm

The underlying fundamentals behind QDS flux calculation remain constant regardless of whether or not the solver employs the local Graphics Processing Units. The QDS algorithm consists of three basic steps:

1. Generation of a small number of representative particles N (typically 3-4 per coordinate direction) which are used to carry fluxes of mass, momentum and energy between cells. The amount of mass and velocity of these representative particles are drawn from the Maxwellian distribution of velocities by approximating this distribution by the weights and abscissas of a Gauss-Hermite quadrature. For a spatially first order accurate simulation, each particles masses, velocities and internal energies are [1]:

$$m_{ij} = \frac{\rho_i \Delta x w_j}{\sqrt{\pi}}$$

$$v_{ij} = u_i + \sqrt{2\sigma_{vi}^2}\, q_j$$

$$\varepsilon_{ij} = \frac{(\xi - \Omega)\sigma_{vi}^2}{2}$$

   where Δx is the uniform grid size, ζ is the total number of degrees of freedom and Ω is the number of simulated translation degrees of freedom (for example, one dimensional simulations use $\Omega = 1$). The values of w_j and q_j are the weights and abscissas of the Guass-Hermite quadrature [9]. For $N = 3$, these are approximately $w_j = \{1.18163, 0.2954, 0.2954\}$ with corresponding abscissas of $q_j = \{0, 1.2247, -1.2247\}$. The quantities of density (ρ_i), velocity (u_i), velocity variance (σ^2_{vi}) and energy (E_i) are cell properties.
2. Calculation of fluxes between cells over the computational time step Δt. During this time, fluxes are calculating assuming free molecular flight in the same way as conventional kinetic theory based schemes. When a regular, Cartesian grid is employed, no knowledge is required of the neighbouring cell locations or properties – these fluxes are a function of the source cell alone.
3. Exchange fluxes of mass, momentum and energy between cells. Calculate each cells equilibrium macroscopic properties (i.e. density, temperature, bulk velocity) using the updated values.

A flowchart describing the calculation procedure is provided in Figure 1. The main strengths of the QDS algorithm are (i) the entire simulation procedure (minus initialisation and post-processing) are performed on the GPU device, and (ii) the calculation of fluxes from each source cell can be easily performed on separate threads. During the flux calculation procedure, no communication is required between cells. This communication is required only when the fluxes are exchanged between cells. The current implementation employs a second order spatial accuracy, meaning that neighbouring cell information is required for the calculation of the local gradients in each cell used for higher order flux calculation, the details of which can be found in [1].

Validation and Performance

1D Shock Tube Test

A standard test for the compressible Euler Equations is Sod's 1D shock tube. The Riemann analytical continuum solution for a shock tube allows the properties of the flow structure, including the shock propagation velocity W, the contact surface velocity u_P along with the pressure, temperature and density, to be determined at any given time [10]. The simulations were conducted using an ideal monatomic gas. The end walls were simulated as reflective walls. The initial conditions in the high pressure and low pressure ends of the shock tube

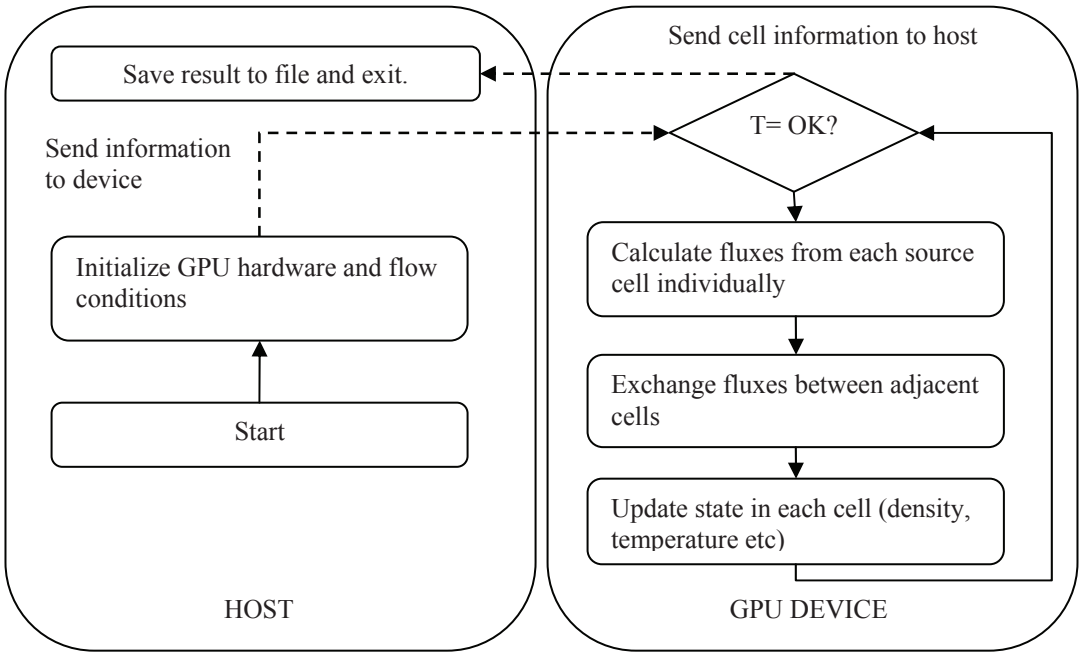


Figure 1: Basic flow diagram showing application of QDS using GPU. The entire calculation procedure occurs on the GPU device with communication between the host and device only occurring prior to and after successful completion of the simulation.

are $p_H = 10p_L$ and the temperatures at both ends of the tube are the same. At t = 0 the infinitely thin diaphragm separating the two gases at $x = 0.5L$ is removed. The resulting propagating shock wave ($M_S = 1.55$) travels from the high pressure region into the low pressure region. The simulation is performed on both single CPU and employing the GPU hardware specified in Table 1. The results obtained are identical in each case and both are in sound agreement with the analytical solution, as shown in Figure 2. The CPU time required by each code is presented in Table 2. As the number of cells increases, disadvantages associated with device initialisation diminish and the GPU capable code quickly demonstrates speed-ups of over 30 times when compared to the single CPU code.

2D/3D Blast Wave Simulation in Urban Environment

Following the previous application of TDEFM to the simulation of blast waves in urban environments [11] we apply the QDS solver to two dimensional (and later, three dimensional) simulations to the simulation of shocked gas flow in urban environments. Some sample results are shown in Figure 3 with computational times and speed-ups demonstrated in Table 3. As expected, speed-ups continue to increase with problem-size. The times shown are for two dimensional simulations due to the difficulty of performing large 3D simulations on single CPU systems.

Two dimensional results (not shown) are in agreement with existing TDEFM and direction-coupled EFM simulations.

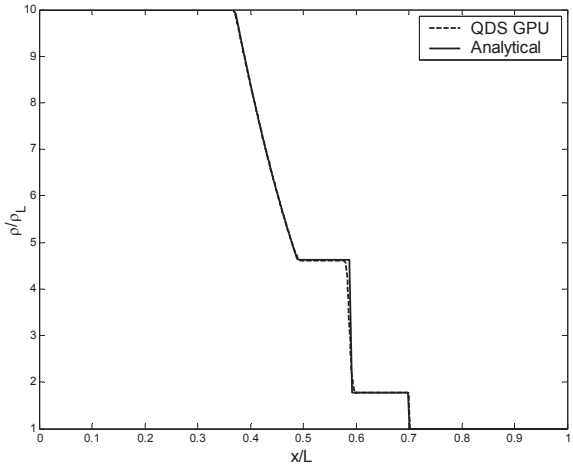


Figure 2: Normalised density from both the analytical solution [ref] and 2nd order accurate QDS at flow-time $T(RT)^{0.5}/L = 0.2$ using 1000 cells. The initial pressure ratio across the diaphragm placed at $x = 0.5L$ was $P_H/P_L = 10.0$ with a uniform temperature throughout. The gas is assumed ideal and monatomic.

Hardware	Details
CPU	Intel Xeon quad-core X5472, 3.0 GHz clock, L2 cache = 12MB.
GPU Hardware	NVidia Tesla S1070 GPU Computing Server (4 x Nvidia Tesla T10 GPU's each with 1.44 GHz clock, 4GB DDR3 Ram, 240 cores) Capable of single or double precision.

Table 1: Details of computer hardware used to compile and run QDS simulations. The operating system used was openSUSE 10.2.

Code	CPU time (milliseconds) vs. Number of Cells						
	256	**512**	**1024**	**10000**	**20000**	**40000**	**50000**
CPU	12.8	51.2	203.2	24167	110519	467783	742792
CPU+T10 (GPU)	14.2	29.2	60.1	1197	3715	13063	20125
Speedup	**0.9x**	**~1.7x**	**~3.4x**	**~20x**	**~29x**	**~35x**	**~37x**

Table 2: Details of CPU times required by both single CPU and GPU capable QDS codes using the computer hardware specified in Table 1. The specified CPU times are the total run times and include the GPU hardware initialisation times and time required to write results to local hard drive.

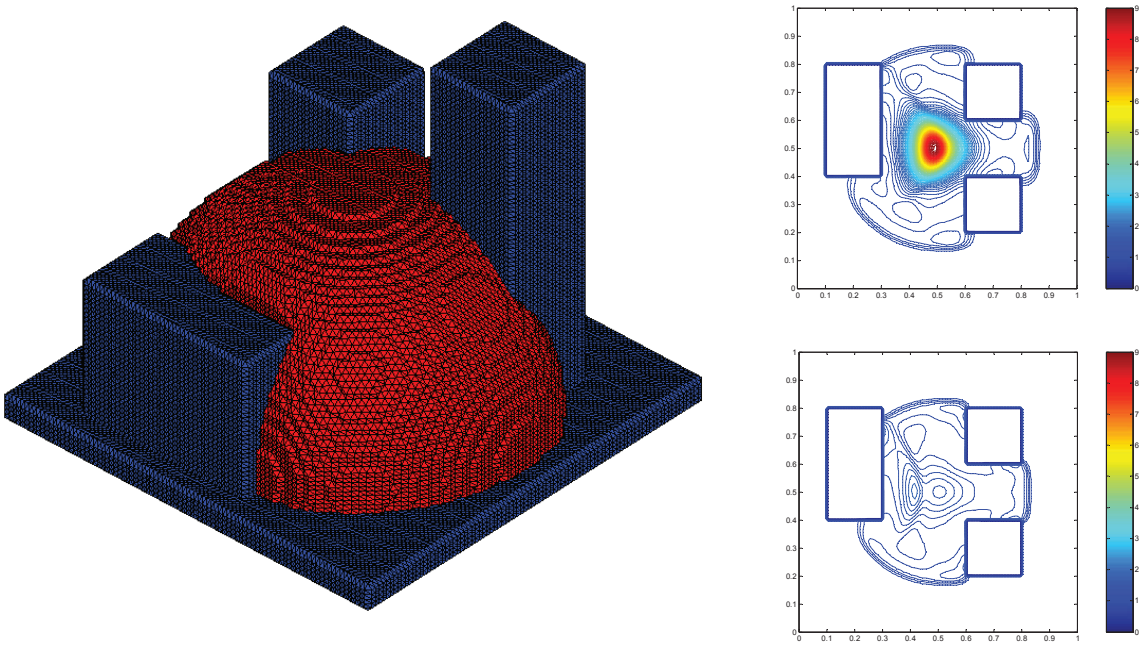


Figure 3: 3D Blast wave simulation in an urban environment using 2nd order accurate QDS at flow-time $T(RT)^{0.5}/L = 0.1$ using 1000000 cells. The "bomb" is modelled using a high temperature region with $T_B/To = 100$ in the region bounded by ($0.45L < x < 0.55L$), ($0.45H<x<55H$). (Left) 3D rendered image of the buildings (in blue) and the location of the propagating shock wave front (in red), (Right, Top) Contours of temperature at ground level, and (Right, Bottom) Contours of temperatures at 0.3H above ground level.

Code	CPU time (seconds) vs. Number of Cells			
	128x128	**256x256**	**512x512**	**1024x1024**
CPU	10.3	88.5	718.3	5970
CPU+T10 (GPU)	0.3	2.2	16.3	124
Speedup	**~34x**	**~40x**	**~45x**	**~48x**

Table 3: Details of CPU times required by both single CPU and GPU capable QDS codes using the computer hardware specified in Table 1 for 2D simulation of blast waves in urban environments. The specified CPU times are the total run times and include the GPU hardware initialisation times and time required to write results to local hard drive.

Conclusion

Presented is the Quiet Direct Simulation (QDS) method slightly modified for and applied to calculation using Graphics Processing Units (GPU's). The simplicity of the QDS algorithm, which requires no evaluation of complex functions (only addition, subtraction, multiplication and division are used), makes the code not only very fast but also suitable for calculation on GPUs. Presented are results for a simple one dimensional benchmark test calculated with codes using both single CPU and multiple CPU (GPU) implementations. The presented results demonstrate that QDS, when combined with GPU computation technology, offers a significant speed up (> 30 times) when compared to conventional, single CPU simulations. This allows the future possibility of very large scale simulations to be run on relatively small (and cheap) equipment without the requirement for traditional supercomputing clusters.

Acknowledgements

The computing facilities and financial support provided by the National Centre for High Performance Computing (NCHC) in HsinChu, Taiwan, is greatly appreciated. In addition, the financial and academic support provided by National Chiao Tung University (NCTU, Taiwan) is also appreciated.

References

1. M.R. Smith, H.M. Cave, Y.-S. Chen, M.C. Jermy and J.-S. Wu, An Improved Quiet Direct Simulation Method for Eulerian Fluids Using a Second-Order Scheme, *J. Comput. Phys.* 228, 2213-2224 (2009).
2. T. Brandvik and G. Pullan, Acceleration of a 3D Euler Solver using commodity graphics hardware, in *46th AIAA Aerospace Sciences Meeting and Exhibit* (2008).
3. E. Elsen, P. LeGresley and E. Darve, Large calculation of the flow over a hypersonic vehicle using a GPU, *J. Comput. Phys.* 10148-10161 (2008).
4. T.R. Hagen, J.M. and J.R. Natvig, Solving the equation equations on Graphics Processing Units, in *International Conference on Computing Science,* (4), 220-227, (2006).
5. M.R. Smith, M.N. Macrossan and M.M. Abdel-Jawad, Effects of direction decoupling in flux calculation in finite volume solvers, *J. Comput. Phys.* 227(8), 4142-4161 (2008).
6. D.I. Pullin, Direct simulation methods for compressible ideal gas flow, *J. Comput. Phys.* 34, 231-144 (1980).
7. B.J. Albright, W. Daughton, D.S. Lemons, D. Winske, and M.E. Jones, Quiet direct simulation of plasmas, *Phys. Plasma* 9(5), 1898-1904 (2002).
8. B. J. Albright, D.S. Lemons, M.E. Jones, and D. Winske, Quiet direct simulation of Eulerian fluids, *Physical Review E* 65, 1-4 (2002).
9. W.H. Beyer, CRC Standard Mathematical Tables, (CRC Press, Boca Raton (FL), 1987).
10. E. F. Toro, Riemann solvers and numerical methods for fluid dynamics: A practical introduction, Springer-Verlag Berlin (1999).
11. M.R. Smith, H.M. Cave, J.-S. Wu and A. Ferguson, Simulation of debris formation and movement resulting from a blast wave in an urban environment, in *15th National Taiwan CFD Conference (Best Paper Award), Kaohsiung, Taiwan,* (2008).

Heterogeneous Parallelism of High-Order Residual Distribution Schemes Using Central and Graphics Processing Units

Stephen M. J. Guzik and Clinton P. T. Groth

University of Toronto Institute for Aerospace Studies, 4925 Dufferin Street, Toronto, ON, M3H 5T6, Canada
Email: *sguzik@utias.utoronto.ca, groth@utias.utoronto.ca*

Abstract

The parallelism of a high-order residual-distribution algorithm is investigated on a heterogeneous system consisting of a multi-core CPU and a graphics processing unit (GPU). Solutions of the steady-state Euler equations are obtained by executing a logistically simple, but computationally expensive, portion of the algorithm on the GPU while the logistically complex, but relatively inexpensive, remainder is simultaneously computed on the CPU cores. The resulting hybrid parallelism, combined with the massive efficiency of the GPU, conceals the computationally expensive portion and the overall speedup is shown to be defined by the fraction executing on the CPU cores, according to Amdahl's law. Based on the observed speedup, the monetary savings provided by a heterogeneous CPU and GPU architecture over that of only CPU cores are estimated.

1 Introduction

The potential for using graphics processing units (GPU) to assist with high performance computation (HPC) has recently generated considerable interest. The GPU devotes more transistors to data processing than the CPU, which diverts significant resources to data caching and flow control [1]. While the CPU can efficiently process conditional instructions (branching) and dispersed data, the GPU is specialized for computation of highly parallel data using instructions with a high arithmetic intensity. Programs that exhibit these characteristics can benefit from the massive computational power of the GPU over the CPU. Perhaps most importantly, the GPU is a very cost-effective parallel processor since research and development are supported by a large graphics visualization market.

In this work, the potential of using GPUs to assist with the computation of discrete solutions to systems of partial differential equations using high-order residual-distribution ($\mathcal{RD}$) techniques is explored. The $\mathcal{RD}$ method is an attractive candidate for GPU computing because it provides a compact stencil (lowering memory operations) and the high-order extension increases the arithmetic intensity. The resulting algorithm illustrates the use of several levels of parallelism including a heterogeneous level featuring simultaneous processing by the CPU and GPU.

2 $\mathcal{RD}$ Methods for the Euler Equations

Residual-distribution methods are cell-vertex methods that are usually solved on simplexes (triangles in two space dimensions). In the following analysis, numerical solutions are obtained for the two-dimensional Euler hyperbolic system of conservation laws given by

$$\frac{\partial \mathbf{U}}{\partial t} + \vec{\nabla} \cdot \vec{\mathbf{F}} = 0\,, \tag{1}$$

with

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_T \end{bmatrix}, \ \mathbf{F}_x = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho u h_T \end{bmatrix}, \ \mathbf{F}_y = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho v h_T \end{bmatrix}, \tag{2}$$

where e_T is the specific total energy and h_T is the specific total enthalpy. Gaseous flows of air are considered and the preceding partial differential equations are supplemented with the ideal gas law, $p = \rho RT$, as an equation of state. The specific gas constant, R, is taken to be $287\ \mathrm{J}/(\mathrm{kg} \cdot \mathrm{K})$ and a perfect gas is assumed with a specific heat ratio of $\gamma = 1.4$.

The numerical solution is obtained in an iterative manner by calculating the residual (or fluctuation) on an element, E, of an unstructured mesh and then, by some appropriate technique, distributing the fluctuation to the vertices of that element to advance the solution in time. The residual-distribution method can be summarized in three steps: computation of the fluctuation, distribution of the fluctuation, and evolution of the solution.

The details of the $\mathcal{RD}$ method are available elsewhere [2, 3]. However, it is necessary to recognize the three steps listed above, and also that two quadratures that must be performed: integration of the fluctuation in each element by

$$\boldsymbol{\phi}^E = -\oint_{\partial E} \vec{\mathbf{F}} \cdot \hat{n}\, d\mathcal{S}\,, \tag{3}$$

and integration of the linearized state in primitive variables, $\mathbf{V} = [\rho, u, v, p]^T$, by

$$\bar{\mathbf{V}} = \frac{1}{\Omega_E} \int_E \mathbf{V}\, d\Omega_E\,. \tag{4}$$

3 High-Order Method

Fourth-order solutions are obtained by following a framework similar to that of finite-element theory. The concept was first adapted to $\mathcal{RD}$ methods by Abgrall and Roe [4]. As illustrated in Fig. 1, P^3 elements are defined from an ordered collection of *primary* elements, enabling construction of a polynomial with 10 degrees of freedom (vertices).

The high-order method only alters the integration of the fluctuation and the linearized state. The distribution of the fluctuation and the evolution of the solution are the same in the primary elements as for standard second-order schemes. Because the solution is represented by cubic polynomials of the primitive state variables, the flux on each edge of an iso-parametric element can potentially be a polynomial of degree 14, requiring eight Gauss points. Integration of the linearized state requires four Gauss points in each element.

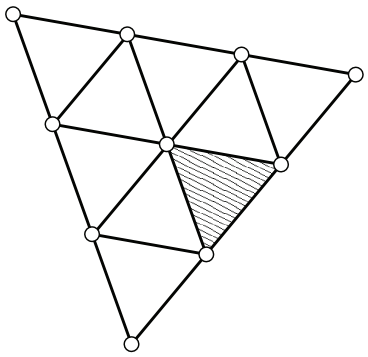


Figure 1: P^3 element consisting of an ordered collection of primary elements (shaded).

4 Parallel Implementation

The target architecture for this research is a distributed HPC cluster with an arbitrary number of identical nodes. Each node is assumed to contain a number of CPU cores sharing random-access memory (RAM) and a GPU with its own dedicated RAM. Communication between the nodes is accomplished using MPI. Within each node, the parallelism between the CPU cores is expressed using POSIX threads. As well, a single POSIX thread is used to control processing

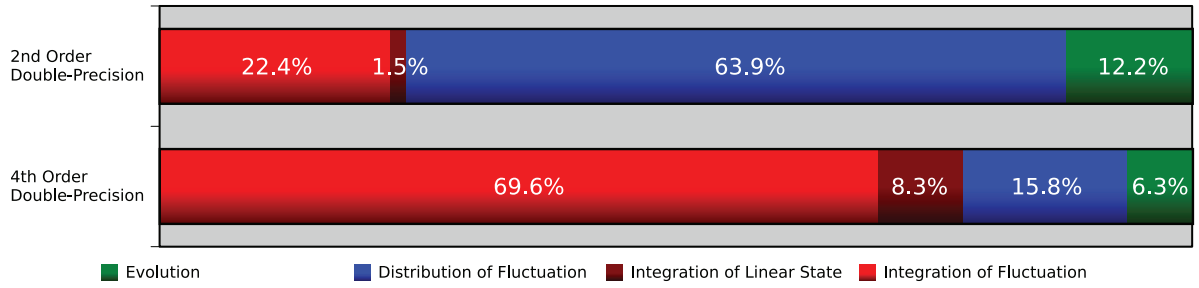


Figure 2: Profile of the fourth-order $\mathcal{RD}$ algorithm for a solution of Ringleb's flow on a quad-core CPU.

on the GPU. The remainder of this paper focuses only on the parallel interaction between the CPU cores and the GPU within a single node. Specifically, our test node consists of a quad-core AMD Phenom II 940 CPU and an NVIDIA GTX 260 (27 multiprocessors and 896 MB) GPU.

The numerical implementation that we envision features several levels of parallelism. The grid is divided into zones and the coarsest level of parallelism is defined by the set of zones distributed to each node. Within a node, the zones in the set are distributed among the CPU cores. The finest levels of parallelism are expressed by the vector processing of the GPU. The details of the parallelism of the GPU are described in the next section. However, there is also an intermediate level of parallelism which is characterized by the simultaneous processing of the GPU and CPU cores.

For parallel execution at the intermediate level, the algorithm is divided depending on which processor a portion of the algorithm is best suited for. Attractive candidates for GPU processing are the quadratures of the fluctuation and linearized state. The imposition of fourth-order accuracy provides a high arithmetic intensity and the compactness of the $\mathcal{RD}$ stencil (9 elements in Fig. 1) provides highly parallel data. Figure 2 shows a profile of the $\mathcal{RD}$ algorithm executing on the quad-core CPU. The computational cost of the quadratures increases dramatically when switching from second-order accuracy to fourth-order accuracy. Precise timings of the algorithm show that 72.4 % of the time is devoted to integration in the fourth-order algorithm.

During each iteration of the algorithm, a streaming process is defined where the GPU integrates the fluctuation and linear state in a set of zones before passing the zones to the CPU cores. The CPU cores then distribute the fluctuation while the GPU simultaneously begins integration of the next set of zones.

5 Quadrature on the GPU

The finest levels of parallelism occurs on the GPU. A complete description of the GPU architecture is available in [1]. Here, a brief description is provided to illustrate how the quadratures are decomposed to execute on the GPU. Current NVIDIA hardware (e.g., GeForce 8, 9, and 200 series) are composed of a set of *multiprocessors*. Each multiprocessor features a single-instruction, multiple-data (SIMD) architecture with 8 processors that each execute a single instruction on different data. The various NVIDIA GPUs generally only differ by the number of available multiprocessors.

The programming of the quadrature is closely aligned with the hardware of the GPU. Blocks are defined which are distributed to the multiprocessors of the GPU. GPU threads are defined which execute in parallel on the processors following the SIMD pattern. A block is prescribed by two P^3 Lagrange elements and 144 threads are used per Lagrange element to perform the integration. As stated in section 3, eight Gauss points are required on each edge for integration

of the flux and four Gauss point in the interior for integration of the linearized state. During integration of the linearized state, 16 threads are used in each element to simultaneously evaluate all Gauss points for all variables. During integration of the flux, 8 threads simultaneously evaluate all Gauss points on the edge for a single variable. This is repeated four times for each equation in the Euler system of equations.

6 Results

Ringleb's flow, a hodograph solution to the Euler equations that involves an isentropic and irrotational flow contained between two streamlines, was used as a test case to evaluate the parallel performance. In Fig. 3 Mach contours show a transonic flow and the partitioning of the mesh into six zones is illustrated. The mesh was constructed using Gmsh [5] and partitioned using the spectral partitioning algorithm of Chaco [6].

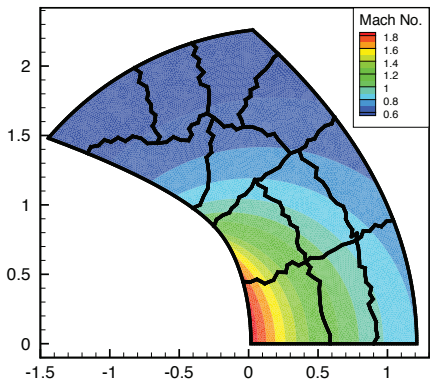


Figure 3: Mach contours for Ringleb's flow. The mesh has been partitioned into twelve zones.

With the entire computation in double precision, the addition of the GPU provides a speedup of approximately 3 times over computation on the two CPU cores alone. According to Amdahl's law, the maximum speedup that can be expected by parallelization of the integration in the results of Fig. 2 is $1/(1-0.72) = 3.6$.

The synchronization and processor utilization is illustrated in Fig. 4 for a single iteration of the algorithm. Zones are numbered in Fig. 4 as they stream between the processors. Integration of the linearized state and fluctuation (red) is performed for four zones at once on the GPU. The results are transferred (yellow) to the CPU cores so that distribution of the fluctuation can be performed (blue). Before a CPU core computes the distribution, the solution for the next set of zones is packaged (magenta) and sent to the GPU (orange). Thus, the four CPU cores and the GPU process concurrently. Because global time stepping is used, the solution cannot be evolved until the residual has been distributed for all zones, and the global time step is known. As soon as the CPU cores evolve the first set of zones, the GPU is tasked to integrate these zones for the *next* iteration while the CPU cores continue to evolve the solution (green) for the remaining zones in the *current* iteration. Processing of the zones on the CPU cores starts with the light blue sections which indicate time spent configuring the *workspace* on the CPU. This includes such tasks as computing additional variable states and applying boundary conditions. Note that the workspace setup was not isolated from the distribution of the fluctuation in Fig. 2. It is shown here because the results from the GPU are not requested until just before they are needed (after workspace setup but before distribution of the fluctuation). The load balancing of the four CPU cores is not perfectly even because of extra computations associated with boundary conditions in some zones.

It is interesting to consider the capital costs of the hardware. The GPU costs about the same as the quad-core CPU. With a speedup of 3, equivalent computing on only CPUs would cost 1.5 times as much. Similar savings could be expected for operational (electricity) costs.

The GPU appears fairly occupied in Fig. 4 but the hardware is still being underutilized. The results shown herein are all computed with double precision floating-point numbers and while this helps ensure the accuracy of the solutions, the single-precision computational power of the GPU is entirely neglected. The theoretical single-precision performance of the GTX 260 is 12 times that of the double-precision performance; use of single-precision should be maximized.

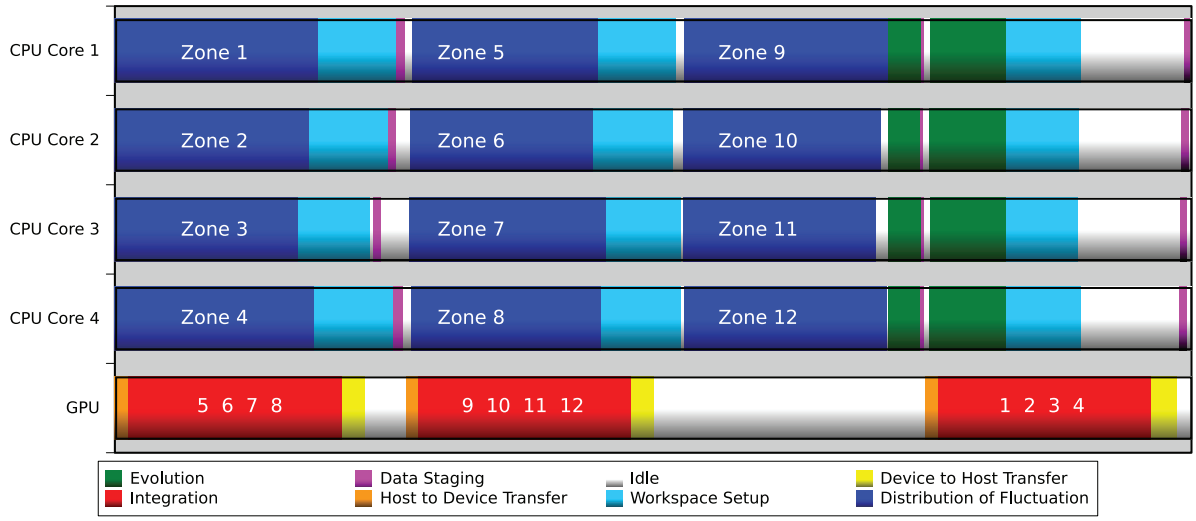


Figure 4: Processor synchronization and usage for one iteration of the fourth-order $\mathcal{RD}$ algorithm.

Experiments have revealed that full single-precision $\mathcal{RD}$ computations of Ringleb's flow do not yield satisfactory accuracy. However, it is probable that some portions can be performed in single precision to better utilize the GPU. At the very least, the initial transient solution can be largely solved in single-precision and the steady-state solution refined in double precision.

7 Conclusions

Usage of a GPU to assist computation of a fourth-order $\mathcal{RD}$ algorithm applied to the Euler equations was found to yield a speedup of approximately three times. Equivalent computing with only CPUs is estimated to have a monetary cost of 1.5 times as much. Better results should be expected if parts of the $\mathcal{RD}$ algorithm could be solved in single precision.

Our own experience with programming the GPU using CUDA is encouraging. In this study, it was shown that a significant speedup could be achieved by only porting a small, but computationally expensive, section of the algorithm to the GPU. As a coprocessor, the GPU can be effectively utilized to enhance the computational efficiency of CFD algorithms.

References

[1] NVIDIA Corporation, *NVIDIA CUDA Programming Guide*, November 2007, Version 1.1.

[2] Deconinck, H., Sermeus, K., and Abgrall, R., AIAA 2000-2328, Fluids Conference, 2000.

[3] Guzik, S. M. J. and Groth, C. P. T., *International Journal of Computational Fluid Dynamics*, Vol. 22, No. 1–2, 2008, pp. 61–83.

[4] Abgrall, R. and Roe, P. L., *Journal of Scientific Computing*, Vol. 19, No. 1-3, December 2003, pp. 3–36.

[5] Geuzaine, C. and Remacle, J.-F., "Gmsh" *International Journal for Numerical Methods in Engineering*, 2009, Submitted.

[6] Hendrickson, B. and Leland, R., "Chaco User's Guide" SAND 94–2692, Sandia, 1994.

TURBULENCE

Three-Dimensional Parallel Adaptive Mesh Refinement Simulations of Shock-Driven Turbulent Mixing in Plane and Converging Geometries

***Manuel Lombardini*[1] *and Ralf Deiterding*[2]**

[1]Graduate Aeronautical Laboratories, California Institute of Technology, Pasadena, CA 91125, USA

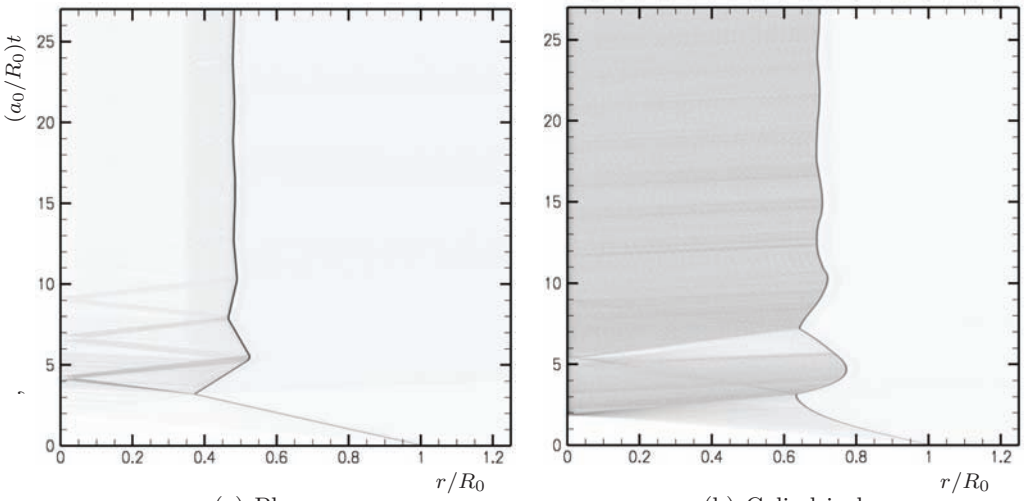
[2]Oak Ridge National Laboratory, P.O. Box 2008 MS6367, Oak Ridge, TN 37831, USA

Key words: Structured Adaptive Mesh Refinement, Large-Eddy Simulations, Richtmyer-Meshkov Instability, Compressible Flows, Turbulent Mixing

This work presents on-going research on the large-eddy simulations (LES) of the Richtmyer-Meshkov instability (RMI) in both plane and converging geometries and the turbulent mixing generated by this flow. The RMI occurs when a perturbed interface between two fluids of different density is accelerated impulsively by a shock wave depositing baroclinic vorticity at the interface. Examples of the occurrence of the RMI are present in experiments aiming to achieve inertial confinement fusion, in natural phenomena such as supernova collapse, or in technologies involving supersonic combustion.

The reference problem consists of an air/SF_6 plane interface in a light-to-heavy configuration, initially impacted by a planar incident shock of Mach number M_I and then reshocked after reflection of the transmitted shock off the endwall of the shocktube [1]. Besides, a canonical simulation of the RMI in a cylindrical converging geometry (90° wedge) has been set up in parallel to validation experiments of converging shocks in a wedge currently conducted by the group of Prof. Dimotakis at GALCIT. An imploding cylindrical shock impacts a perturbed, cylindrically-shaped density interface that separates light air (outside) from heavy SF_6 (inside). The transmitted shock converges down the wedge, reflects off the z-axis, and reshocks the interface, initiating a strong turbulent mixing, similar to the plane case. A study of the wave diagrams describing the shock-contact interaction with reshock shows differences between the two geometries, as presented in Fig. 1. In particular, multiple reshocks follow the first reshock in the converging geometry, while expansion wave reverberations are observed in the plane geometry.

The reshock process produces a large dynamical range of turbulent scales, necessitating the use of LES. While conventional LES models generally consider only resolved-scale transport and do not attempt to capture the mixing process between the two fluids, the stretched-vortex subgrid-scale (SGS) model of Misra & Pullin, extended to compressible flows by Kosovic *et al.* and subgrid scalar transport by Pullin, is based on an explicit structural modeling of small-scale dynamics (see [2] for detailed references). This is an SGS model that utilizes stretching vortices as the essential subgrid element in the closure of Favre-filtered Navier-Stokes equations by providing the subgrid stress tensor τ_{ij}, the turbulent temperature flux q_i^T, and the mixture fraction flux q_i^ψ. This model enables



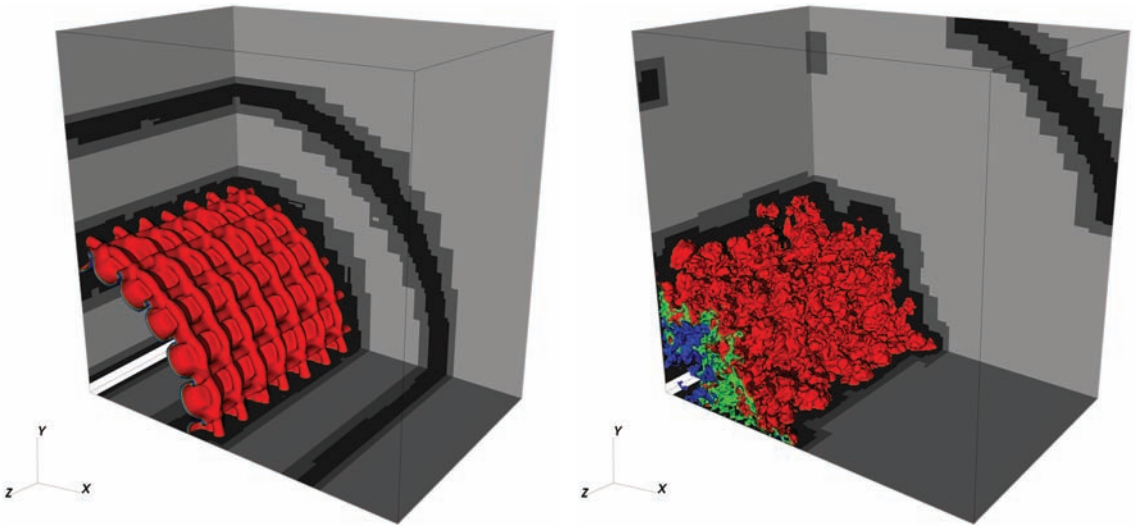
(a) Plane (b) Cylindrical

Figure 1: Wave diagrams for light-to-heavy air/SF_6 $M_I = 1.2$-shock interactions

a computational paradigm for multi-scale LES that extends estimates of some turbulent statistics from the resolved cutoff-scale to the Kolmorogov and Batchelor scales.

The resolution requirements imposed by the flow physics vary greatly both spatially and temporally for these simulations. For example, different key features such as shock waves of different strengths and turbulent mixing regions (as seen on Fig. 2 and 3) need more resolution than other smoother regions of the flow. This is provided presently through LES within the AMROC framework developed by Deiterding [3] and based on the structured adaptive mesh refinement algorithm (SAMR) of Berger & Oliger [4]. Discrete conservation of mass, momentum, and energy is accomplished by using a flux-based conservative finite-difference approach. To illustrate the need of AMR, consider our larger converging cylindrical simulation ($M_I = 2.0$ incident cylindrical shock as it impacts the cylindrical interface), for which the domain is discretized with $95 \times 95 \times 64$ cells on the base grid with three additional levels of refinement based on the local density gradient. The refinement ratio between each level is equal to 2 for all levels and directions, and the subgrid cutoff scale is set to that of the finest mesh. As the flow evolves, the distribution of the AMR hierarchy to different processors is adjusted dynamically to balance the work and all parallel data structures are automatically rearranged (cf. Fig. 2). The simulation was performed using 32 AMD Opteron 2.5 GHz-quad-processor nodes (16 GB memory each) and consumed about 70,000 h CPU time. The cell count varies from a minimum of approximately 10 million cells in the early times of the simulation, when the mixing zone has not yet radially expanded, to a peak of around 140 million at late times. AMR reduces the computational expenses compared to the equivalent finest unigrid $760 \times 760 \times 512$ problem that would have used approximately up to 3 times more storage and taken more than 3 times longer to complete.

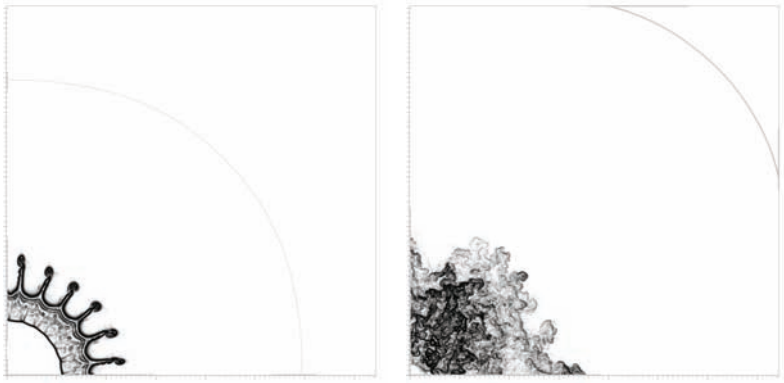
The numerical method is formulated for Cartesian uniform grids, and is applied to each subgrid of the mesh hierarchy. It is an extension of the hybrid method by Hill & Pullin [5] to SAMR meshes. A weighted, essentially non-oscillatory (WENO) scheme is used to capture discontinuities (such as shock waves or fine/coarse mesh interfaces) but switches to a low-numerical dissipation, explicit, tuned center-difference scheme (TCD) in the smooth or turbulent regions, optimal for the functioning of explicit LES such as the SGS stretched-vortex method. To ensure discrete numerical stability of the inviscid terms (momentum,



(a) (b)

Figure 2: Light-to-heavy air/SF_6 $M_I = 2.0$-converging cylindrical RMI: Differently colored iso-surfaces corresponding to mass fractions $\psi = 75\%$, 50%, and 25% visualize the evolution of the mixing zone (a) after the first imploding shock interaction and (b) after the first exploding reshock. The gray levels on the background planes represent the domains of different mesh refinement (only first 3 levels displayed)

scalar and energy convection terms), the centered discretization are written in a stable, energy preserving (skew-symmetric) formulation adapted to compressible flows [6]. For the subgrid activity to be correctly computed, thereby assuring the quality of the LES, the use of WENO is restrained to regions containing shock waves only. Switching between WENO and TCD has been optimized using a detection criterion based on Lax's entropy condition [7].



(a) (b)

Figure 3: Light-to-heavy air/SF_6 $M_I = 2.0$-converging cylindrical RMI: azimuthal view (plane slice $z = 0$) of Schlieren density gradient photographs showing shock waves and density stratifications (a) after the first imploding shock interaction and (b) after the first exploding reshock (all 4 levels used)

The parallelization strategy implemented in AMROC is a rigorous domain decomposition approach. Accumulating the work of all refinement levels, the root level is split at runtime utilizing a Hilbert space-filling curve (cf. [3]). All higher levels follow the base level whenever the distribution to processors changes (operation "Recomposition"). Subgrids are synchronized with layers of ghost cells that are also used to implement physical and coarse-fine interface boundary conditions. The operation "Boundary value setting" is dominated by the overhead involved with parallel data sychronization. As a scalability test for AMROC we consider a 3-level simulation of a confined explosion problem using the WENO scheme. The time per base level iteration of the most expensive operations is displayed in Fig. 4(a). As it can be expected with the chosen parallelization approach, linear speed-up is achieved for the block-based numerical update of the finite volume scheme ("Integration"), while the scaling of the operations "Boundary value setting" and "Recomposition", that involve parallel communication, is less optimal. We have found the parallel performance of the overall algorithm (cf. Fig. 4(b)) suitable for effective parallel computations on several hundred processors.

A detailed quantitative analysis of results has been obtained from post-processing on saved parallel data files that are read in on smaller CPU count. The investigation includes space-time histories of instantaneous plane/cylindrical shell-averages $\langle . \rangle$ of diverse quantities Q, taken parallel/concentrically to the main shocks. Typically, we need to evaluate shell-averages of about a hundred base quantities ($\langle \rho \rangle$, $\langle \rho^2 \rangle$, etc.) across the entire domain. For each time considered, the post-processing required around 1,000 h CPU. From the base shell-averages, we define, for example in the cylindrical geometry:

$$\begin{aligned} Q(\boldsymbol{x},t) &\equiv \langle Q \rangle(r,t) + Q'(r,\theta,z,t), \\ \widetilde{Q}(r,t) &\equiv \frac{\langle \rho\, Q \rangle}{\langle \rho \rangle}, \\ \mathrm{Var}_\rho(Q)(r,t) &\equiv \widetilde{Q^2} - \tilde{Q}^2 = \frac{\langle \rho Q^2 \rangle}{\langle \rho \rangle} - \frac{\langle \rho Q \rangle^2}{\langle \rho \rangle^2}, \end{aligned}$$

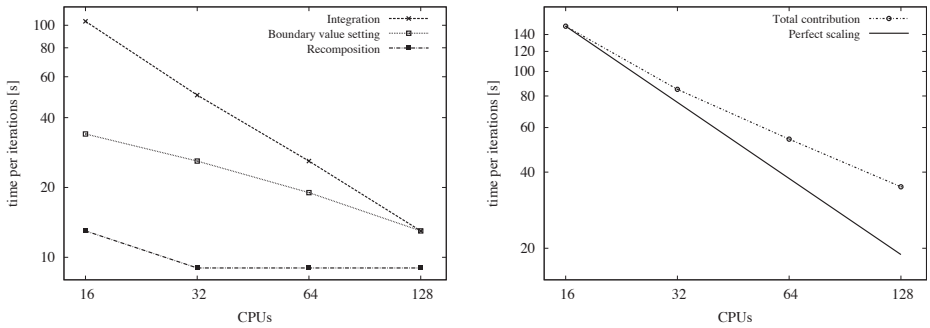
where ρ is the total gas density, and $\mathrm{Var}_\rho(Q)$ is the Favre-like variance of Q. The analysis illustrates the fundamental role of the subgrid model in the turbulent dissipation, the turbulent kinetic energy (as shown in Fig. 5), the scalar dissipation, etc. In addition, from the general Navier-Stokes equations, shell-averaged equations are derived for the turbulent kinetic energy, the turbulent mass flux in the axial/radial direction, and the density variance. These statistics are ultimately used to estimate the relative importance of each term involved in the shell-averaged equations. This approach represents a first step to understanding and eventually modeling the turbulent mixing characteristics of such complex shock-driven flows in both geometries.

Diverse power spectra, including velocity components, density, and scalar spectra, as well as other quantities such as Taylor and Kolmogorov microscales are evaluated within the turbulent mixing zone. Additionally, mixing properties across the width of the mixing zone are investigated via the Reynolds joint density-mixture fraction probability density function:

$$\tilde{\mathrm{P}}(\psi;\boldsymbol{x},t) \equiv \frac{1}{\langle \rho \rangle} \int \rho \mathrm{P}(\rho,\psi;\boldsymbol{x},t) \mathrm{d}\rho,$$

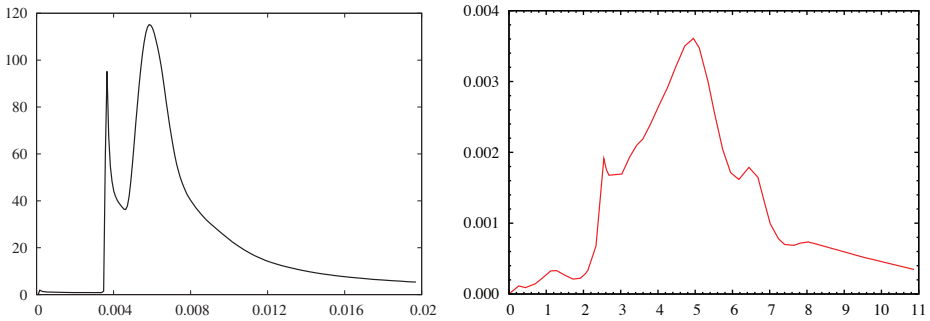
where ψ is the mass fraction.

The dicussion of the numerical results will emphasize the similarities and differences between both geometries and finally allow us to consider the present AMR-LES framework as



(a) (b)

Figure 4: CPU scalability test: 3D adaptive computation with 3 grid levels (coarse grid of $30 \times 30 \times 30$ cells, uniformly refined grid of $120 \times 120 \times 120$ cells (1.73 Mcells)) of a circular shock-wave expanding in an enclosed box. The reflected shocks interact in a complex manner. (a) represents the breakdown of computational costs in second per iteration, (b) the total time per iteration



(a) (b)

Figure 5: Light-to-heavy air/SF_6 $M_I = 1.2$-RMI: Total volume-averaged TKE vs. time for the (a) plane and (b) cylindrical geometries (arbitrary units)

a credible tool for the detailed study of practical multi-scale applications involving turbulent compressible flows.

[1] M. Vetter and B. Sturtevant, "Experiments on the Richtmyer-Meshkov instability of an air/SF_6 interface," Shock Waves **4**, 247-252 (1995).

[2] D. J. Hill, C. Pantano, and D. I. Pullin, "Large-eddy simulation and multiscale modeling of a Richtmyer-Meshkov instability with reshock," J. Fluid Mech. **557**, 29-61 (2006).

[3] R. Deiterding, "Construction and application of an AMR algorithm for distributed memory computers," In Adaptive Mesh Refinement – Theory and Applications, T. Plewa, T. Linde, and V. G. Weirs, Eds., vol 41 of Lecture Notes in Computational Science and Engineering, 361-372 (2005).

[4] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial-differential equations," J. Comput. Phys. **53**, 484-512 (1984).

[5] D. J. Hill and D. I. Pullin, "Hybrid tuned center-difference-WENO method for large-eddy simulation in the presence of strong shocks," J. Comput. Phys. **194**, 435-450 (2004).

[6] A. E. Honein and P. Moin, "Higher entropy conservation and numerical stability of compressible turbulence simulations," J. Comput. Phys. **201**, 531-545 (2004).

[7] M. Lombardini, "Richtmyer-Meshkov instability in converging geometries," PhD thesis , Caltech (2008).

Large Scale Simulation of Turbulence using a Hybrid Spectral/Finite Difference solver

J. Bodart*, L. Joly*, J.-B. Cazalbou*

** Université de Toulouse, Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)*
(Tel : +335-61-33-91-82; e-mail: authorname@isae.fr)

Abstract:

Performing Direct Numerical Simulation (DNS) on large scale systems (offering more than 10^4 cores) has become a challenge in high performance computing. Successful attempts have been made recently using fully spectral codes with a 2D domain decomposition, referred to as the volumetric decomposition (See for example [6, 10]). Compared to the former slabwise decomposition, this method, initiated by Eleftheriou *et al.* [7], the number of cores usable for a problem of size N^3 increases from N to N^2. This highly scalable way to perform three dimensional FFT has been improved and implemented recently in the Open Source library *P3dfft*, by Pekurovsky [9] at the San Diego Supercomputer Center. An other advantage of the 2D decomposition is the degree of freedom added in the grid management. It gives the opportunity to use unequal number of grid points (N_x, N_y, N_z), without affecting the overall scalability [5]. Thus the grid can be chosen in better agreement with the physics of the flow. We propose to take advantage of the volumetric decomposition in situations where one direction is treated with finite-differences schemes. Such schemes give the possibility i/to solve non-periodic boundary directions encountered in jet or channel flows for instance ii/to use nonuniform grids to study non-homogeneous turbulence (mixing layers and aforementioned situations), while keeping the spectral accuracy in two directions.

The coupling between the possible number of grid points in each directions and the processor distribution is studied and tested on the Blue Gene/P architecture. Scalability of the algorithm is presented using up to 32768 cores. Finally, preliminary results of a complete simulation using 10^9 grid points on 16384 cores is presented. From the authors knowledge it is the largest DNS simulation using i/only two homogeneous directions ii/a volumetric decomposition in a hybrid spectral/finite difference solver. In this situation where the turbulence is not fully homogeneous, 2.5 millions of CPU hours are required to obtain converged statistics.

Keywords: Turbulence, Large Scale DNS, Blue Gene/P

1. FLOW CONFIGURATION UNDER STUDY

With the aim of investigating the interaction between turbulence and a solid wall,we have designed a configuration, where the turbulence self diffuses from a plane source towards a rigid wall as presented Figure 1. The turbulence production does not rely on the presence of mean-shear but it is instead synthesized inside a bounded layer using a specific forcing field. The effects induced by the presence of a mean shear thus cancel. The viscous effects as well as the kinematic blocking can be fully isolated in the near wall region as first demonstrated by Campagne et al. [2]. The resulting turbulent flow field is homogeneous with respect to x and y and rotation invariant about the z axis, which justify the implementation of a Navier Stokes solver mixing finite-difference and spectral schemes. The x and y directions are treated using the same number of Fourier nodes N_{xy}, while the z direction uses a different number of grid points N_z and a non-uniform distribution. The forcing field is implemented as a source term in the Navier Stokes equations.

$$\frac{\partial u_i}{\partial t}+u_j\frac{\partial u_i}{\partial x_j}=-\frac{1}{\rho}\frac{\partial P}{\partial x_i}+\nu\frac{\partial^2 u_i}{\partial x_j\partial x_j}+f_i(x,t) \quad (1)$$

It is nonzero only in the central region of the domain (dark-blue on Figure 1). It has been designed in the physical space to satisfy several constraints which are mainly i/random in time ii/divergence-free iii/localized in space. This kind of forcing demonstrates good abilities to synthesize a plane source of turbulence [1].

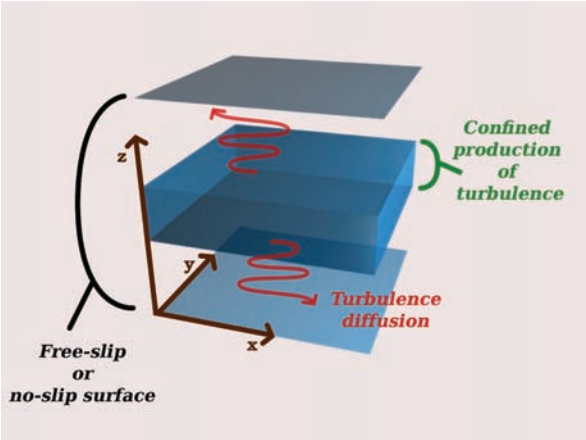


Fig 1: *Numerical set up*

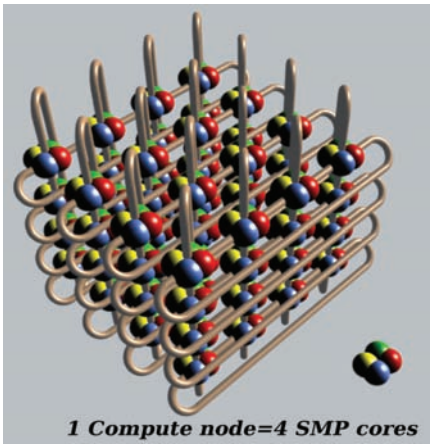


Fig 2: *3D torus network on Blue Gene/P*

The computations are started with a flow at rest. During a transient, the amount of turbulent energy contained in the whole domain is growing under the action of the forcing field. A statistically steady state is reached when the power induced by $f(x,t)$ is statistically balanced by the viscous dissipation. This numerical setup can be seen as an analogue of the oscillating-grid experiments (see for example [4]).

2. PARALLEL PERFORMANCES OF THE NAVIER STOKES SOLVER

Navier-Stokes equations are solved in the hybrid Fourier/physical space. The present solver is pseudo-spectral, based on Fourier modes in the two periodic directions, and on finite differences with a sixth-order compact scheme [8] in the wall-normal direction. The parallel implementation is performed in the same way as in a fully spectral solver. In the latter, the parallel efficiency is strongly conditioned by the three dimensional FFT. This algorithm requires 2 or 3 different data distributions with the following characteristics: the arrays are distributed over one or two directions in such a way that 3D data with the same array index in the third direction is kept: i/local on a CPU core and ii/contiguous in memory. Thus, the FFT computation in each direction and/or the finite difference resolution is performed locally on a single core and does not require any parallel communications. Therefore, implementation efforts concentrate on the data transposition from one domain to an other, in order to minimize the cost of the communications involved. The slabwise decomposition involves a single transposition and is the most efficient implementation in many cases. With this method however, a problem of size N^3 can be distributed over N cores only, which is a strong limitation on nowadays supercomputers.

Recently, Pekurovsky developed the open-source library *p3dfft* to perform 3D FFT using a volumetric decomposition, meaning that data are distributed over two instead of a single direction, thus involving $n = p \cdot q$ cores. This library, written in fortran calls optimized FFT libraries such as ESSL or FFTW. Its scalability has been demonstrated on Blue Gene/L [5]. We use a slightly modified version in our C code to perform 2D FFT only and solve finite-difference equations with the optimized Blas/Lapack libraries at the last stage If $p = q = N$ the number of usable cores is extended from N to N^2 for a problem of size N^3. In Figure 3 the data distribution is performed using 4 cores ($p = q = 2$) marked with different colors. We observe that yellow/green and blue/red blocks constitute two independent groups of cores where the parallel communications remain internal during the first transposition. The n MPI processes define a virtual p x q 2D grid where communications are performed independently inside each line/column during the first/second transposition. The Blue Gene/P physical topology is a 3 dimensional torus (X, Y, Z) of quad-cores processors, where each processor can directly communicate with its nearest neighbor in each of the three directions, as presented in Figure 2. In the fully distributed mode (VN), the 4 SMP cores generate a fourth dimension in the topology (X, Y, Z, T), where the communications times are the fastest. To achieve the best performance during the computation of 2D FFT, two classes of parameters need to be set to their optimal values:

- The couple (p, q).
- The mapping of a virtual process 2D topology on the physical cores.

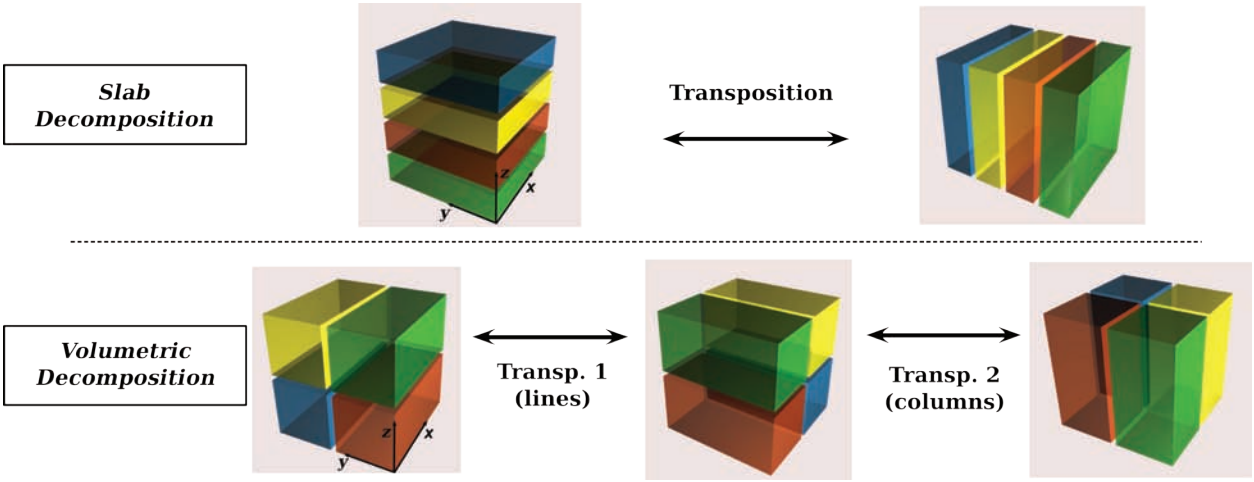


Fig 3: *Slab decomposition versus volumetric decomposition for the FFT computation.*

Using the VN mode, selecting the predefined mapping *TXYZ* and maximizing the slowest varying index *p* in the processor grid gives the best results. In this optimal configuration, processors are numbered using the T direction in the 4D torus as the fastest varying index. Different 2D mappings were tested on a static configuration, with negligibly smaller time restitution obtained in best cases. In fact, building such a 2D optimized mapping is worthwhile in situations where the parallel communications are mostly of neighbor to neighbor type. However, the semi-global communications involved by the transpositions induce many communication types and reach their maximum efficiency in the predefined mapping available on the Blue Gene/P.

3.LOAD BALANCING

Classically, the data packing chosen in *p3dfft* transform an array from $N_x \cdot N_y \cdot N_z$ points in the physical space to $(N_x+2)/2 \cdot N_y \cdot Nz$ Fourier modes in the spectral space (assuming that N_x is even). This data packing introduces a slight load unbalance, as an optimal distribution of the N_x points over p processors in the physical space will usually result of an unbalanced and non optimal distribution of the $(N_x+2)/2$ Fourier modes over q processors in the spectral space. Although this is not crucial in terms of data exchange between processors, it affects the loop sizes when working locally on a processor at the finite-difference stage. The related extra cost can reach 10% in worse cases. The phase-shift dealiasing method used for the nonlinear terms has to be coupled with a spherical truncation [3] which removes the aforementioned modes $k((N_x+2)/2, y, z)$. Thus we ignore this modes and avoid unnecessary operations, while keeping the desired accuracy.

4. GRID DEFINITION

Using the volumetric decomposition, we gain a degree of freedom in the grid generation. It gives the possibility to use different grid points numbers in different directions, contrasting with the N^3 size imposed by the slabwise decomposition. The grid definition has to fulfill constraints of different kinds in homogeneous/inhomogeneous directions. In the homogeneous direction, it is mandatory to satisfy the spatial decorrelation and resolve the smallest uniform length scale, whereas in the second case the main issue is to map the grid according to the smallest length scale. The new degree of freedom brought by relaxing the constraint $N_z=N_x=N_y$ is particularly useful when the calculation domain should minimize the CPU cost generated by the transient, or when the different directions of the problem are decoupled such as in jet flows. In our configuration the total number of grid points is set as a function of N_{xy} and N_z, corresponding to the dimension x or y, and z. The volumetric decomposition yields:

$$N_{tot} = N_{xy}^2 N_z = N_{xy} \frac{N_{xy}}{p} \frac{N_z}{q} pq \quad (2)$$

$N_{xy} = k_1 \cdot p$ and $N_z = k_2 \cdot q$ are therefore two necessary conditions of optimal load balancing. This freedom degree, which decreases with the number of cores, is not limited to large scale simulations. Figure 4 shows all (N_{xy},N_z) couples ensuring optimal load balancing for both decompositions if the number of cores is set to $n=1024$ or $n=16384$.

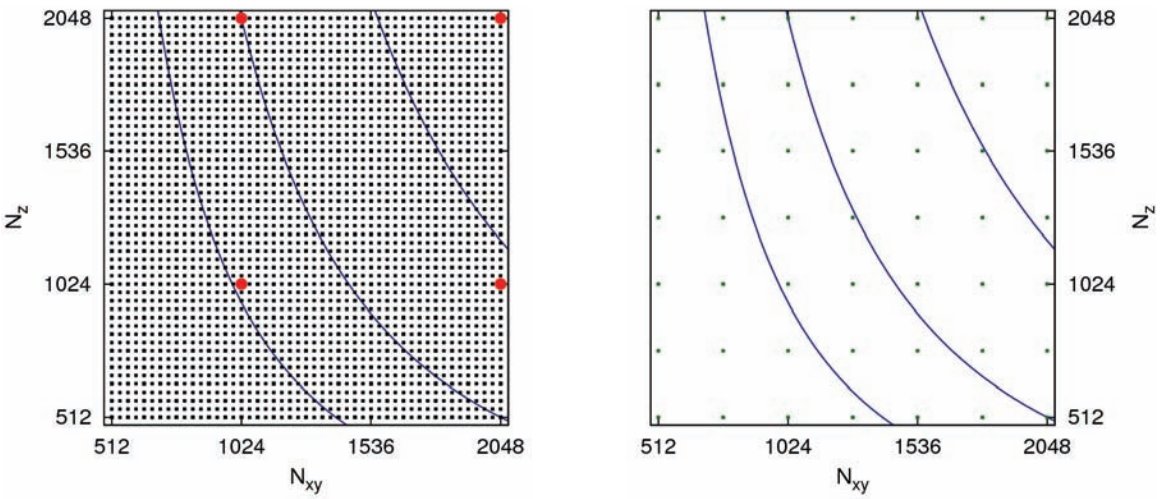


Fig 4: *Possible combinations of (N_{xy},N_z) when using the volumetric decomposition:*
left- *For n = 1K cores,* right - *For n = 16K cores. Red circles indicate the slabwise decomposition (p=1). Blue lines indicate isovalues of the total number of grid points.*

In the present computation, the domain is cubic with the forced layer filling one third of the domain. The forced-eddy size is set to one quarter of the horizontal dimension. These characteristics ensure the spatial decorrelation in the homogeneous directions and a sufficient diffusion length. The box size being set, we introduce a method to estimate the anisotropy induced by the rigid wall at the small scales based on a similarity hypothesis with intermediate length scales, leading to the definition of two "oriented" Kolmogorov length scales η_{xy} and η_z. The number of points in the homogeneous direction is adapted to satisfy the resolution criterion $(k_{xy})_{Max}.\eta_{xy}=1.5$. The mapping of the last direction follows the same rule using the modified wave number k'_z, which is adequate with our sixth-order compact scheme. More grid points are required to resolve the z direction with this domain size and a targeted Reynolds number, justifying the choice of a *1024* x*1024* x*1280* grid.

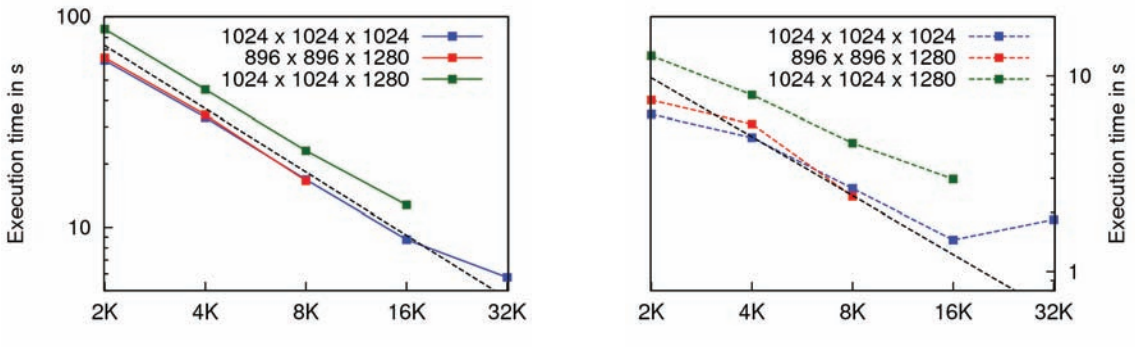


Fig:5: *Parallel scaling (lK=1024cores):* left-*Overall scaling,* right-*2D FFT scaling*

CONCLUSION AND ONGOING WORK

The volumetric decomposition is shown to be particularly adapted to mixed spectral/finite-difference solver as demonstrated by the large scale scalability test presented on Figure 5. The degree of freedom provided by the 2D decomposition gives a flexibility close to full finite-differences/volumes solvers, and offers an excellent overall scalability with a spectral accuracy in two directions. The final paper will present the currently running *1024* x*1024* x*1280* test case, and preliminary results which includes first and second order statistics as well as Reynolds-stresses tensor budgets.

ACKNOWLEDGMENT

Preliminary computations were carried out under the HPC-EUROPA++ project (project number: 1187), with the support of the European Community - Research Infrastructure Action of the FP7 "Coordination and support action" Programme.

Final computation is being carried out on the Blue Gene/P supercomputer at IDRIS (Paris).

REFERENCES

1. Bodart, J., Joly, L. and Cazalbou., J.-B. (2009) Local large scale of unsheared turbulence. *Direct and Large Eddy Simulation 7*, Trieste (Italy), in press
2. Campagne, G., Cazalbou, J.-B., Joly, L. and Chassaing, P. (2006) Direct numerical simulation of the interaction between unsheared turbulence and a free-slip surface. *ECCOMAS CFD 2006*, TU Delft, (The Netherlands)
3. Canuto, C, Hussaini, M. Y., Quarteroni, A. and Zang, T.A. (1988) *Spectral methods in Fluid Dynamics*. Springer-Verlag.
4. De Silva, I. P. D. and Fernando, H. J. S. (1994) Oscillating grids as a source of nearly isotropic turbulence. *Physics of Fluids*, 6(7):2455-2464.
5. Donzis, D. A., Yeung, P. K. and Pekurovsky, D. (2008) Turbulence simulations on o(10^4) processors. *Teragrid'08 meeting*, Las Vegas, Nevada.
6. Donzis, D. A., Yeung, P. K. and Sreenivasan, K. R. (2008) Dissipation and enstrophy in isotropic turbulence: Resolution effects and scaling in direct numerical simulations. *Physics of Fluids*, 20(4)
7. Eleftheriou, M., Moreira, J. E., Fitch, B. G. and Germain, R. S. (2003) A Volumetric FFT for Blue-Gene/L. *High Performance Computing – HiPC*, Hyderabad (India), p 194-203.
8. Lele, S. K. (1992) Compact finite difference schemes with spectral-like resolution. *Journal of Computational Physics*, 103(1):16-42.
9. Pekurovsky, D. http://www.sdsc.edu/us/resources/p3dfft/index.php.
10. Schumacher, J. and Putz,M. (2007) Turbulence in laterally extended systems. *Parallel Computing, Architectures, Algorithms and Applications*, vol 38, p 585-592.

Turbulent flow around a wall-mounted cube: direct numerical simulation and regularization modelling

F. X Trias[a,b], A. Gorobets[a], R.W.C.P. Verstappen[b], M. Soria[a], A. Oliva[a]

[a] *Centre Tecnològic de Transferència de Calor, Technical University of Catalonia*
ETSEIAT, c/ Colom 11, 08222 Terrassa, Spain, E-mail: cttc@cttc.upc.edu
[b] *Institute of Mathematics and Computing Science, University of Groningen*
P.O. Box 407, 9700 AK Groningen, The Netherlands, E-mail: R.W.C.P.Verstappen@rug.nl

Abstract

Since direct numerical simulation (DNS) cannot be performed at high *Re*-numbers, a dynamically less complex mathematical formulation is sought. In the quest for such formulation, we consider regularization (smooth approximations) of the nonlinearity. The regularization method basically alters the convective terms to reduce the production of small scales of motion by means of vortex stretching. In doing so, we propose to preserve the symmetry and conservation properties of the convective terms exactly. This requirement yields a novel class of regularizations that restrain the convective production of smaller and smaller scales of motion by means of vortex stretching in an unconditional stable manner, meaning that the velocity can not blow up in the energy-norm (in 2D also: enstrophy-norm). The numerical algorithm used to solve the governing equations preserves the symmetry and conservation properties too. In the present work, regularization modelling is tested for a fully-3D geometry: turbulent flow around a wall-mounted cube at $Re_h = 7235$ (based on the cube height and the bulk velocity). Modelled results are compared with the new DNS results carried out on the MareNostrum supercomputer using 300 CPU on a structured Cartesian mesh with $\sim 16 \times 10^6$ points. The algorithm used to solve the Poisson equation works well on arbitrarily meshed 3D grids and is therefore well-suited for the proposed DNS simulation. Moreover, details about the Poisson solver for DNS and the parallelization of code are also discussed.

Key words:
DNS, regularization modelling, wall-mounted cube, parallel 3D Poisson solver, symmetry-preserving discretization

1. Introduction

Turbulent flow around a wall-mounted cube has been the subject of numerous experimental and numerical studies in the last two decades. Results showed that this flow is mainly characterised by the appearance of a horseshoe-type vortex at the upstream face, an arc-shaped vortex in the wake of the cube, flow separation at the top and side faces of the cube and vortex shedding. However, most of the numerical studies have been performed using RANS and LES modelling techniques while accurate DNS simulations are quite scarce and limited to very low Reynolds number (the reader is referred to the review by [1] and the references therein). Moreover, since this flow configuration is used for benchmarking purposes to validate turbulence models and numerical methods the availability of DNS results at relatively high Reynolds numbers is of extreme importance. Thus, a new complete DNS simulation at $Re_h = 7235$ (based on the cube height and the bulk velocity) has been performed on MareNostrum supercomputer using 300 CPUs. To do so, a new parallel Poisson solver [2] for fully-3D parallelepipedic geometries has been used. However, at high *Re*-number, DNS simulations are not feasible.

In this context, a dynamically less complex mathematical formulation is sought. In the quest for such a formulation, we consider regularizations (smooth approximations) of the nonlinear convective term. The first outstanding approach in this direction goes back to Leray [3]; the Navier-Stokes-α model also forms an example of regularization modelling (see [4], for instance). The regularization methods basically alter the convective terms to reduce the production of small scales of motion. In doing so, we proposed to preserve the symmetry and conservation properties of the convective terms exactly [5]. This requirement yielded a family of *symmetry-preserving regularization* models [5]: a novel class of regularizations that restrain the convective production of smaller and smaller scales of motion in an unconditional stable manner, meaning that the velocity can not blow up in the energy-norm (in 2D also: enstrophy-norm). The numerical algorithm used to solve the governing equations preserves the conservation properties too [6] and is therefore well-suited to test the proposed simulation model.

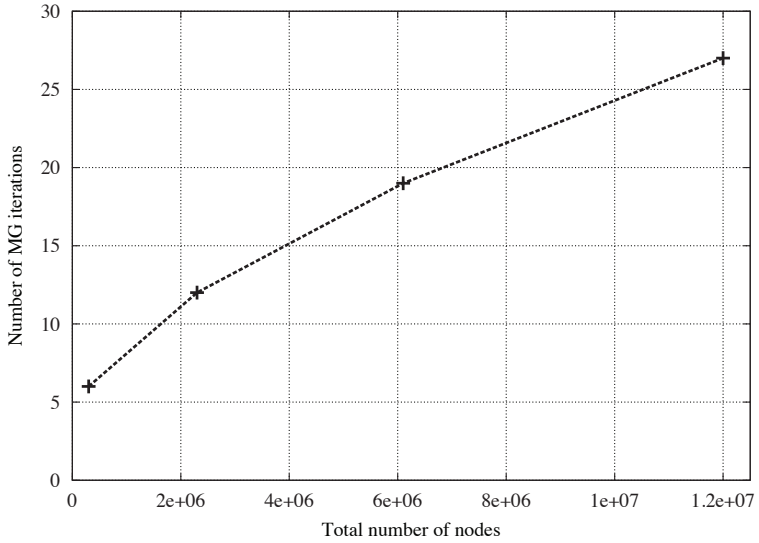


Figure 1: Scalability test of the 3D Poisson solver: number of MG iterations with mesh growth

2. Governing equations and numerical methods for DNS

The non-dimensional incompressible Navier-Stokes equations in a parallelepipedic domain $\Omega = (0, L_x) \times (0, L_y) \times (0, L_z) \subset \mathbb{R}^3$ in primitive variables are considered

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = \frac{1}{Re}\Delta \mathbf{u} - \nabla p \; ; \qquad \nabla \cdot \mathbf{u} = 0 \tag{1}$$

where Re is the non-dimensional Reynolds number. Equations (1) are discretized on a staggered grid in space by symmetry-preserving schemes [6]. For the temporal discretization, a fully explicit dynamic second-order one-leg scheme is used for both convective and diffusive terms. Finally, to solve the pressure-velocity coupling a classical fractional step projection method is used. Further details about the time-integration method can be found in [7]. Regarding the Poisson solver, a novel approach for fully-3D parallelepipedic geometries has been used. The solver, named MG-KSFD, is based on a two-level Multigrid (MG) method. As a second-level solver we use KSFD [8], a solver for 3D problems with one uniformly meshed periodic direction. Then, a CG with a local band-LU preconditioner is used as smoother. Scalability test for the DNS simulation described in the following section is displayed in figure 1. For further details about the Poisson solver the reader is referred to our previous work [2].

3. DNS of a wall-mounted cube at $Re_h = 7235$

Turbulent flow around a wall-mounted cube in a channel flow has been chosen as the first demonstrative DNS application for the MG-KSFD solver. Despite the simple geometry, several drawbacks limit the DNS simulation of flows around 3D bluff obstacles:

- Region around the obstacle demands finer grid. Thus, arbitrary meshing is desirable to achieve the required resolution without wasting computational resources in order regions.
- Additionally, the presence of a 3D bluff obstacle itself does not allow to apply directly an FFT-based method (unless some immersed boundary method is used).

The geometry of the wall-mounted cube in a channel is displayed in figure 2 (left). The computational domain is $17h \times 6h \times 3h$ in the streamwise, spanwise and normal to the channel wall directions where h is the cube height. The upstream face of the cube is located at $7h$ from the inlet. For the sake of simplicity[1] the following analytical profile has been prescribed at the inlet

$$U^+ = U/u_\tau = min\left(y^+, k \ln y^+ + B\right) \; ; \qquad V^+ = W^+ = 0 \tag{2}$$

where $y^+ = (y/H)Re_\tau$, $u_\tau = Re_\tau \nu / H$, $k = 0.25$ and $B = 5.0$. H is the channel half-height (in our case $H = 1.5h$). Convective boundary conditions are imposed at the outflow. Global mass conservation is forced through a minor[2] correction of the outflow conditions. Non-slip boundary conditions are imposed at the channel and at the obstacle surfaces. Periodic boundary conditions are imposed in the y-direction.

4. Turbulence modelling: C_4-regularization

Since computational costs of DNS simulations are prohibitive, a dynamically less complex mathematical formulation is needed. In the quest for such formulation, we consider regularizations (smooth approximations) of the non-linear convective term. Here, we restrict ourselves to the C_4 approximation (see [5], for details): the convective term in the Navier-Stokes equations is then replaced by the following $O(\epsilon^4)$-accurate smooth approximation $C_4(u, v)$ given by

$$C_4(u, v) = C(\overline{u}, \overline{v}) + \overline{C(\overline{u}, v')} + \overline{C(u', \overline{v})} \tag{3}$$

where $C(u, v) = (u \cdot \nabla)\, v$ represents the convective operator. Note that here a prime indicates the residual of the filter, *e.g.* $u' = u - \overline{u}$, which can be explicitly evaluated, and $\overline{(\cdot)}$ represents a symmetric linear filter with filter length ϵ. Therefore, the governing equations result to

$$\partial_t u + C_4(u, u) = \frac{1}{Re}\Delta u - \nabla p + f; \qquad \nabla \cdot u = 0 \tag{4}$$

Note that the C_4 approximation is also a skew-symmetric operator like the original convective operator. Hence, the same inviscid invariants -kinetic energy, enstrophy in 2D and helicity- than the original Navier-Stokes equations are preserved for the new set of partial differential equations to be solved (4).

[1] Influence of inflow boundary conditions has been studied. Transient fully-developed channel flow, channel flow averaged profile and analytic profile (2) have been considered. It was found that for cube locations far enough from the inlet ($\gtrsim 5h$, for the range of Re_τ-number studied) no significant differents are observed. Therefore, since the analytical profile is the simplest and easiest-to-reproduce we adopted this inlet boundary conditions.

[2] In practice, several orders of magnitude lower than velocity values.

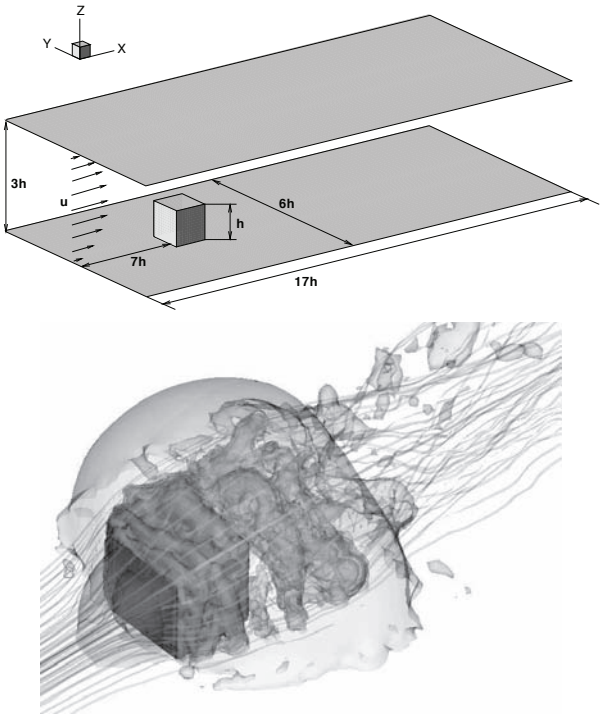


Figure 2: Left: geometry of the wall-mounted cube in a channel. Right: Instantaneous pressure iso-surfaces with streamlines.

4.1. On the dynamic determination of the filter length

Since now the performance of the C_4 approximation has been successfully tested for a relatively wide variety of configurations: a turbulent channel flow [5], a plane impinging jet [9] and an air-filled differentially heated cavity (DHC) of height aspect ratio $A = 4$ at Ra-numbers 10^{10} [10] and 10^{11} [11] by means of direct comparison with the DNS results. In the latter cases, the filter width ϵ was treated as a parameter. Then, the value of ϵ was prescribed in advanced.

In the present work we propose to determine ϵ dynamically with the requirement that the vortex stretching must be stopped at the scale set by the grid. The idea behind is to modify convective operator sufficiently to guarantee that the following inequality is hold

$$\lambda_k(\epsilon) - \nu k^2 \leq 0 \tag{5}$$

where $\lambda_k(\epsilon) = \omega_k \cdot C_4(\omega, u)/(\omega_k \cdot \omega_k)$ is the Rayleigh quotient of the vortex-stretching at the grid scale, $k = \pi/h$. In practice, the value of λ_k has been bounded by the largest (positive) eigenvalue of the straintensor $\mathcal{S}$,

$$\lambda_k(\epsilon) \leq f_{4,k}(\epsilon)\ \lambda_{max}(\mathcal{S}) \tag{6}$$

For the C_4-approximation the damping function, $0 < f_{4,k} \leq 1$, at the highest frequency is given by $3\hat{g}_k^2(\epsilon) - 2\hat{g}_k^3(\epsilon)$ (see [5], for details), where $0 < \hat{g}_k(\epsilon) \leq 1$ is the transfer function of the linear filter. Therefore, it suffices that the following inequality be locally hold

$$3\hat{g}_k^2(\epsilon) - 2\hat{g}_k^3(\epsilon) \leq \frac{\nu k^2}{\lambda_{max}(\mathcal{S})} \quad \longrightarrow \quad \hat{g}_k(\epsilon) \quad \longrightarrow \quad \epsilon \tag{7}$$

to guarantee that the vortex-stretching mechanism is stopped at the smallest grid scale.

5. Conclusions

A new complete DNS simulation of a wall-mounted cube on a channel flow at $Re_h = 7235$ has been carried out on the MareNostrum supercomputer using 300 CPUs. To do so, the fully-3D Poisson solver proposed in our previous work [2] has been used on an arbitrarily meshed 3D grid with $\sim 16 \times 10^6$ points.

On the other hand, since DNS simulations are not feasible for real-world applications, a novel turbulence modelling approach has been proposed. It consists on a regularization of the non-linear convective terms with the requirement that the symmetry and conservation properties must be exactly preserved. This yielded a novel class of regularization named *symmetry-preserving regularization* [5]. In the initial approach [5, 9, 10] the value of the filter length, ϵ, was prescribed in advanced. Instead, here we propose a dynamical method to determine ϵ with the requirement that the vortex-stretching mechanism must be stopped at the smallest grid scale. Therefore, the proposed method constitutes a parameter-free turbulence model. Moreover, since no *ad hoc* phenomenological arguments that can not be formally derived for the Navier-Stokes equations are used it suggest that this method should be valid for other configurations.

A direct comparison between the modelled results and the DNS reference results will be presented in the final paper and the conference. Further details about the parallelization of the code and the Poisson solver will be also discussed.

Acknowledgements

This work has been financially supported by the *Ministerio de Educación y Ciencia*, Spain, (Project: "Development of high performance parallel codes for the optimal design of thermal equipments". Contract/grant number ENE2007-67185) and a postdoctoral fellowship *Beatriu de Pinós* (2006 BP-A 10075) by the *Generalitat de Catalunya*.

Calculations have been performed on the IBM MareNostrum supercomputer at the Barcelona Supercomputing Center. The authors thankfully acknowledge this institution.

References

[1] A. Yakhot, T. Anor, H. Liu, and N. Nikitin. Direct numerical simulation of turbulent flow around a wall-mounted cube: spatio-temporal evolution of large-scale vortices. *Journal of Fluid Mechanics*, 566:1–9, 2006.

[2] A. Gorobets, F. X. Trias, M. Soria, C. D. Pérez-Segarra, and A. Oliva. From extruded-2D to fully-3D geometries for DNS: a Multigrid-based extension of the Poisson solver. In *Parallel Computational Fluid Dynamics*, Lyon, France, May 2008. Elsevier.

[3] J. Leray. Sur le movement d'un liquide visqueaux emplissant l'espace. *Acta Mathematica*, 63:193–248, 1934.

[4] B. J. Geurts and D. D. Holm. Regularization modeling for large-eddy simulation. *Physics of Fluids*, 15:L13–L16, 2003.

[5] Roel Verstappen. On restraining the production of small scales of motion in a turbulent channel flow. *Computers and Fluids*, 37:887–897, 2008.

[6] R.W.C.P. Verstappen and A. E. P. Veldman. Symmetry-Preserving Discretization of Turbulent Flow. *Journal of Computational Physics*, 187:343–368, 2003.

[7] F. X. Trias, M. Soria, A. Oliva, and C. D. Pérez-Segarra. Direct numerical simulations of two- and three-dimensional turbulent natural convection flows in a differentially heated cavity of aspect ratio 4. *Journal of Fluid Mechanics*, 586:259–293, 2007.

[8] A. Gorobets, F. X. Trias, M. Soria, and A. Oliva. A scalable parallel Poisson solver for three-dimensional problems with one periodic direction. *Computers and Fluids*, (under revision).

[9] O. Lehmkuhl, F. X. Trias, R. Borrell, and C. D. Pérez-Segarra. Symmetry-preserving regularization modelling of a turbulent plane impinging jet. In *ERCOFTAC WORKSHOP, Direct and Large-Eddy Simulations 7*, Trieste, Italy, September 2008.

[10] F. X. Trias, M. Soria, A. Oliva, and R. W. C. P. Verstappen. Regularization models for the simulation of turbulence in a differentially heated cavity. In *Proceedings of the European Computational Fluid Dynamics Conference (ECCOMAS CFD 2006)*, Egmond aan Zee, The Netherlands, September 2006.

[11] F. X. Trias, R. W. C. P. Verstappen, M. Soria, A. Gorobets, and A. Oliva. Regularization modelling of a turbulent differentially heated cavity at $Ra = 10^{11}$. In *5th European Thermal-Sciences Conference, EUROTHERM 2008*, Eindhoven, The Netherlands, May 2008.

Parallel Simulation of Turbulent Flow Inside an Aspiration Chamber Using Fluent Software

V. U. Zoria*, J. P. Strodtbeck*, J. M. McDonough, K. I. Logachev*****

** Department of Mechanical Engineering, University of Kentucky,*
Lexington, KY 40506-0503, USA
*** Departments of Mechanical Engineering and Mathematics, University of Kentucky,*
Lexington, KY 40506-0503, USA
, * Shukhov Belgorod State Technological University,*
Belgorod, RUSSIA

Abstract: Parallel simulations of air flow inside an aspiration chamber were performed with Fluent CFD software. The mesh for 3-D geometry contains 2.6 million nodes, and computations were carried out with the number of processors NCPU = 1, 2 and 4. Transport of averaged flow quantities was approximated by Reynolds-averaged Navier–Stokes (RANS) equations. The goal of the present study is to analyze the correlation of computational time with number of processors in the particular geometry of an aspiration chamber in order to assess effectiveness of Fluent parallelization in the context of cluster machine architecture.

Keywords: CFD, parallel computing, Fluent, RANS.

1. INTRODUCTION

One of the important scientific and practical problems of industrial ecology is development of aspiration systems with low energy intensity. The main element of the aspiration system is the local ventilation suction unit which is used to trap polluting substances close to the source. In ventilation suction devices of closed type (aspiration chambers) it is necessary to redirect flow of dust aerosols in the aspiration net; that is, to use an aspiration chamber as an area of preliminary separation of air from dust. This requires knowledge of accurate data for air velocity inside the chamber as well as behavior of polluting aerosols inside the flow field. Numerically, this information can be obtained with help from commercial CFD software packages such as Fluent. Precise and reliable simulation requires a large number of grid points which makes it very time consuming. At this point parallelization becomes a crucial element that can significantly reduce required runtime.

2. GOVERNING EQUATIONS

To compute transport of averaged flow quantities in our simulations we use RANS equations with the standard k-ε turbulence model. In the averaging process, the solution variables of the exact Navier–Stokes equations are decomposed into mean and fluctuating components; the velocity decomposition is:

$$u_i = \bar{u}_i + u_i^{'}, \tag{1}$$

where $\overline{u}_i$ and $u_i^{'}$ are the mean and fluctuating velocity components, respectively, (i = 1,2,3). Pressure and other scalar quantities have similar decompositions:

$$\varphi = \bar{\varphi} + \varphi^{'}. \tag{2}$$

These decompositions are substituted into the exact instantaneous continuity and momentum equations, and subsequent time averaging leads to the usual RANS equations. In a Cartesian coordinate system these are:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i) = 0 \tag{3}$$

$$\frac{\partial}{\partial t}(\rho u_i)+\frac{\partial}{\partial x_j}(\rho u_i u_j) = -\frac{\partial p}{\partial x_i}+\frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}-\frac{2}{3}\delta_{ij}\frac{\partial u_l}{\partial x_l}\right)\right]+\frac{\partial}{\partial x_j}(-\rho\overline{u_i' u_j'}) \quad (4)$$

where overbar notation has been suppressed on all but the Reynolds stress terms.

Equations (3) and (4) are the RANS equations; they have the form of the instantaneous Navier–Stokes equations, where the velocities and other solution variables are now represented by time-averaged values. (We note that for the present simulations for which $\rho \equiv$ const, Eq. (3) is replaced with the divergence-free condition, $\nabla \cdot \boldsymbol{U} = 0$, where $\boldsymbol{U} \equiv (u_1, u_2, u_3)^T$, and the divergence on the right hand side of (4) is omitted.)

Additional terms of the form

$$-\rho\overline{u_i' u_j'}$$

appear in the time-averaged equations and bring the effects of turbulence into the formulation. There are no fundamental equations for these terms, but Fluent provides numerous models. Here we employ the so-called "standard" k-ε model (see, e.g., [1,2] for details).

3. MODEL DEFINITION

To perform the simulations a 3-D grid of the aspiration chamber with inlet and outlet was generated using the Fluent grid generator Gambit. Geometrical and physical parameters of the chamber are shown in Fig. 1. The generated 3-D grid displayed in Fig. 2 contains 2.6 million nodes. These are spaced in a non-uniform manner to help in resolving boundary-layer phenomena, as seen in the figure. Although this is formally an unstructured grid, all cells are parallelepipeds, so, in principle, direct conversion to a structural grid is possible.

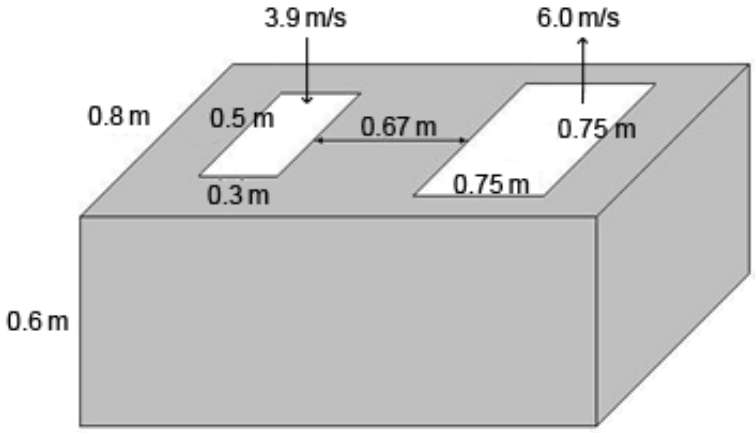


Fig. 1: *Parameters of the aspiration chamber.*

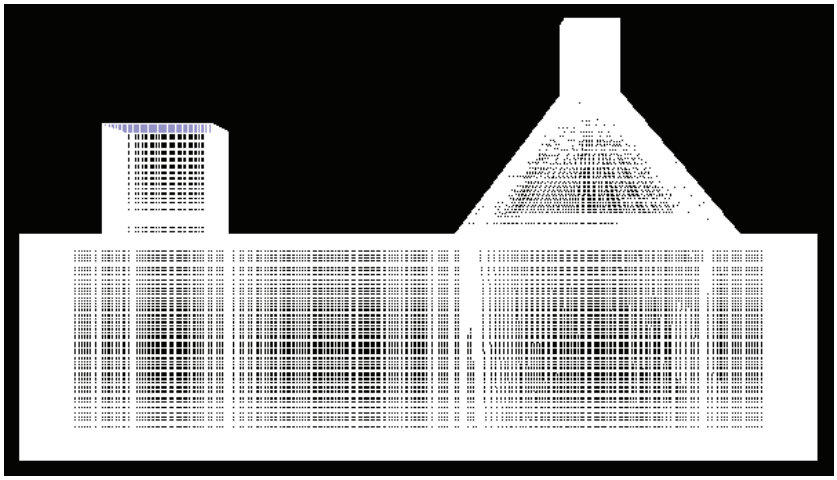


Fig. 2: *Schematic representation of the 3-D grid of the aspiration chamber.*

The inlet velocity is specified by its magnitude normal to the boundary; the fluid is air. The main physical parameters of the simulations are collected in the following table.

Table 1: Flow simulation parameters

Parameter	Numerical Value	Units
Velocity magnitude	3.9	m/s
Turbulent Kinetic Energy	1.0	m^2/s^2
Turbulent Dissipation Rate	1.0	m^2/s^3
Air density	1.225	kg/m^3
Air viscosity	1.7894E-05	kg/m-s

4. PARALLELIZATION IN FLUENT

Fluent's parallel solver allows simulations using multiple processors that may be executing on the same computer, or on different computers in a network. Parallel processing in Fluent involves an interaction between Fluent, a host process, and a set of compute-node processes. Fluent interacts with the host process and the collection of compute nodes using a utility called cortex that manages Fluent's user interface and basic graphical functions. Fluent uses MPI parallelization on both clusters and shared-memory multiprocessor machines, but there is little control available to the user other than specifying a desired number of processors. (See [3] for detailed descriptions.)

5. RESULTS

The computations reported here were carried out on an IBM cluster employing 340 dual-core Intel processors. But only a maximum of 8 of these are available for Fluent applications. Streamlines from the steady flow are shown in Fig. 3, and these are at least in qualitative agreement with physical expectations. In particular, one can observe regions of recirculation in at least some of the expected locations. On the other hand, accuracy, per se, of these calculations is still subject to verification. LES runs are in progress, and these should provide some data against which these RANS calculations can be checked.

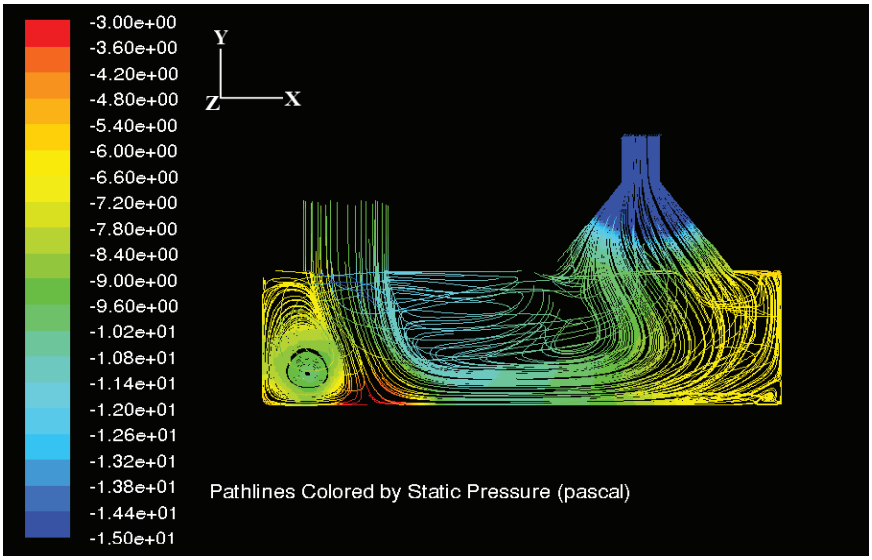


Fig. 3: *Streamlines of air flow inside the chamber.*

Air flow simulation through the aspiration chamber was performed for number of processors NCPU = 1, 2 and 4. These results are summarized in Table 2 in terms of wall-clock time. In Fig. 4 we display the corresponding parallel speed ups.

Table 2: Simulation time vs. the number of processors.

NCPU	Simulation Time
1	45 h, 01 min
2	26 h, 16 min
4	17 h, 41 min

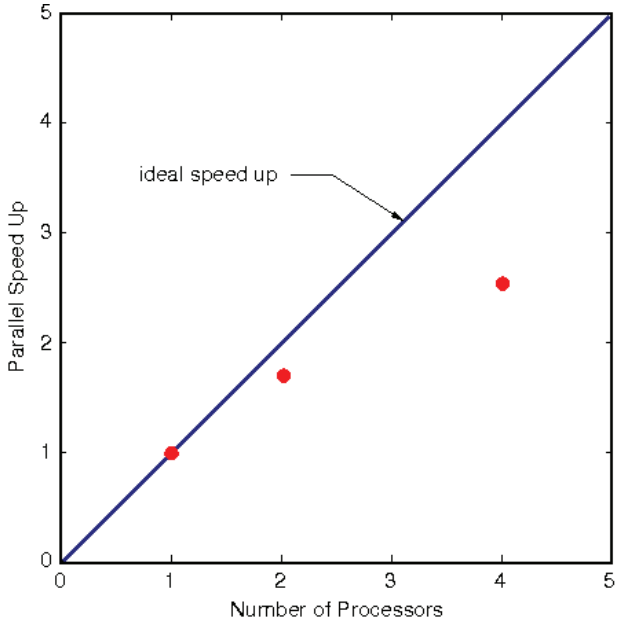


Fig. 4: Fluent speed up.

It is clear that speed ups begin to deteriorate significantly already with only four processors—a result similar to that reported in [4] using open MP with a Fortran 90 code on the same IBM system.
These preliminary results will be further extended in the final paper by using three different mesh sizes to represent the aspiration chamber. The simulations will be done with the number of processors NCPU = 1,2,4 and 8. We will also report results when large eddy simulation (LES) is used as the turbulence modeling technique. We believe this should provide a quite thorough assessment of parallel performance of Fluent for practical problems on IBM clusters.

Acknowledgements: V. Z. wishes to thank the Fulbright International Exchange Program for the great opportunity to conduct scientific research in the United States. The work was done under partial financial support of Russian Foundation for Basic Research (Project 08-08-13687-офи_ц). J. S. acknowledges aid provided by a fellowship from the Kentucky Space Grant Consortium through NASA.

REFERENCES

[1] J. M. McDonough (2008), *Introductory Lectures on Turbulence: Physics, Mathematics and Modeling* http://www.engr.uky.edu/~acfd/lctr-notes634.pdf
[2] D. C. Wilcox (1993), *Turbulence Modeling for CFD*, DCW Industries, Inc., La Canada, CA
[3] *Fluent 6.1 Users' Guide*, http://202.185.100.7/homepage/fluent/html/ug/main_pre.htm
[4] J. M. McDonough and J. Endean (2008), Parallel Simulation of Type IIa Supernovae Explosions, in *Parallel Computational Fluid Dynamics*, 2007, Elsevier, Amsterdam.

ACOUSTICS AND COMBUSTION

Computational Aeroacoustics of a Supersonic Jet Impinging on an Inclined Flat Plate Using High Speed Parallel Computers

Taku Nonomura*, Yoshinori. Goto, Kozo Fujii*****

**University of Tokyo, Sagamihara, Kanagawa 229-8510, Japan*
(Currently ISAS/JAXA, Tel: +81-42-759-8259; e-mail: nonomura@flab.eng.isas.jaxa.jp)
***University of Tokyo, Sagamihara, Kanagawa 229-8510, Japan*
(Currently TOYOTA Motor Company)
****Institute of Space and Astronautical Science/JAXA, Sagamihara, Kanagawa 229-8510, Japan*

Abstract: Aeroacoustic waves from supersonic jet impinging on an inclined flat plate is computed with the high speed parallel computers for improving the prediction of the acoustic waves from rocket plumes. For the parallelization, openMP and MPI techniques are used. From the results, three kinds of acoustic waves are emitted; 1) acoustic waves from the jet-shear layer before impinging, 2) acoustic waves from the impinging, and 3) acoustic waves form the jet-shear layer after impinging. The second kind of acoustic wave has not been well-modeled one for the prediction of the acoustic waves from the rocket plume. This is clarified for the first time with the very high speed parallel computers.

Keywords: computational aeroacoustics, jet acoustics, jet impinging, parallelization, rocket plume.

1. INTRODUCTION

A rocket plume emits very strong acoustic waves, which possibly damage the payload such as artificial satellites or explores. Therefore, the accurate prediction of pressure level of acoustic waves is needed. Thus far, the empirical prediction method presented in NASA-SP8072[1] is used. This empirical prediction method has the strong assumption in which the free-jet acoustic wave sources are located along the predicted jet-path. This leads to its insufficient prediction accuracy for some rocket launch sites.

Computational fluid dynamics (CFD) and computational aeroacoustics (CAA) are now rapidly improving their accuracy due to the high speed computer and high resolution schemes. The use of CFD/CAA will be cure for the prediction of the acoustic waves from a rocket plume. For building the accurate prediction methods for the acoustic waves from a rocket plume, Nonomura and Fujii[2-6] conducted the computations of a supersonic free jet with a high-order shock capturing scheme. They discussed the characteristics of Mach wave sources in supersonic free jet, such as the source position, directivity, frequency characteristics. Now, present authors conducted the CFD/CAA of a supersonic jet impinging on an inclined flat plate, which simplifies the rocket launch configuration. These computations use the large-eddy simulation (LES) for resolving turbulence flow-fields and acoustic wave generation, which requires very fine resolutions. Uses of a high-order shock capturing scheme and 32 to 512 CPUs parallel computation allow us to conduct such a highly resolved computation. In this presentation, the analysis of a supersonic impinging to an inclined flat plate, which is based on the high resolution schemes and highly parallelized computations, is presented.

2. COMPUTATIONAL CONDITIONS

In this analysis, air, of which specific heat ratio γ is 1.4, is used as jet and ambient fluid. The physical quantities are nondimensionalized by the states at the nozzle throat. Following four parameters are chosen to represent the jet-condition; ideally expanded Mach number M_J, designed exit Mach number M_D, Reynolds number based on the nozzle throat diameter *Re,* temperature ratio *TRc* In the present research, *TRc* is defined as the ratio of the chamber temperature to the ambient temperature. In addition, we have two geometric parameters, θ and L/D, where as θ is the angle of inclination of the flat plate, L is the distance from nozzle exit to the flab plate on axis, and D is the nozzle diameter as shown in Fig. 1.

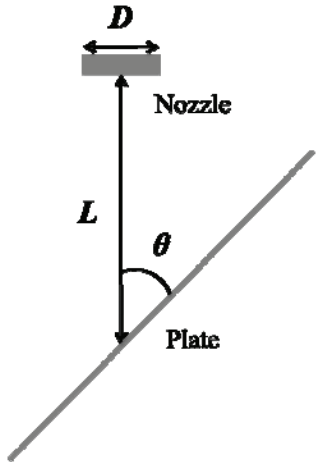


Fig. 1: Definition of geometric parameters

In this paper, the results of the case whose parameters, M_J=2.0, M_D=2.0 Re=10^5, TRc=1 θ=45[deg] and L/D=5.0 is presented. Although Mach number and Reynolds number used in this study are relatively low compared with actual rocket plume condition, they allow us to discuss the characteristics of Mach waves, because the Mach number is high enough to emit the Mach waves and Reynolds number does not have strong effects on Mach wave characteristics except for source positions at the Re>10^5.

3. NUMERICAL METHODS AND PARALLELIZATION

3.1 Numerical Methods

The governing equations are three-dimensional compressive Navier-Stokes equations. Length, density, and velocity are non-dimensionalized respectively by the diameter, the density, and the sound velocity of the nozzle throat.

For the convection term discretization, 7th-order WCNS[7] is employed. In the flux evaluation procedure, SHUS[8] is adopted. WCNS[7,9,10] is a shock capturing high-order scheme which was developed as a combination scheme of WENO and a compact scheme. In this research, the seventh order upwind WENO-like cell-node to cell-center interpolation for conservative variables and the eighth order explicit cell-center to cell-node difference scheme, are used as Nonomura et al. did.[7] A special treatment for computation of metrics proposed by Nonomura et al.[11] is adopted to keep the geometric conservation law. Furthermore, in order to achieve more high resolution, a relative limiter[12] is employed for WCNS smooth indicator. Now, threshold of relative limiter R is selected to be 5. Moreover, a few modifications for achieving high resolution and low computational costs are implemented in WCNS as Reference[13].

For the discretization of viscous term, the sixth-order central difference scheme is employed. Any sub-grid scale models are not used, because the MILES idea is adopted.[14]

The second-order backward differencing is converged by three sub-iterations of ADI-SGS scheme is used for the time-integration, whereas ADI-SGS scheme is based on the idea of theFF-SGS scheme[15]. Time stepping is set to 0.005 which corresponds to 10 for the maximum Courant number, and less than 1 for the Courant number in the jet-shear-layer.

3.2 Computational Grids

The computational region employed in this study is sho*wn in Fig. 2. The buffer region* is set for avoiding non-physical acoustic wave emission or reflections. The present computational grid system is shown in Fig. 3. The whole computational domain is divided into 11 or 128 zones (grid lines in the different 11 zones are drawn with different depth) for grid topology or MPI parallelization as discussed below.

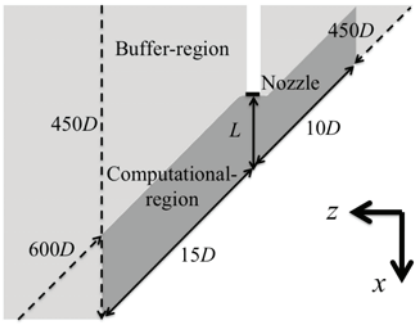


Fig. 2 Computational domain.

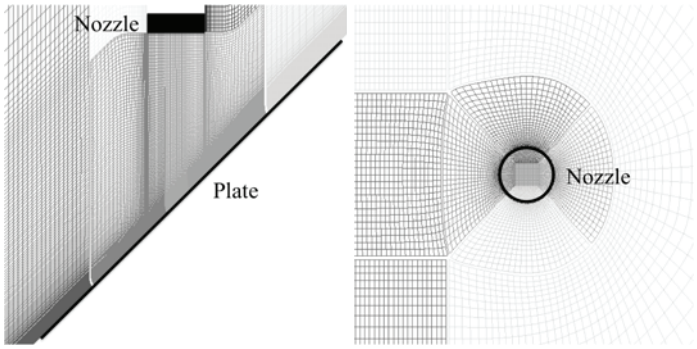


Fig. 3 Computational grids. Left: side view. Right: top view.

3.3 Parallelization

We used SX6(8CPUs/node) by NEC or FX1(4cores/node) by Fujitsu in JAXA. On SX6, 4 process MPI and 8 process openMP parallelizations are conducted, resulting in 32 CPUs parallelization. On FX1, 128 node MPI and 4 process openMP parallelizations are conducted, resulting in 512 cores parallelization. Zonal decomposition is used for MPI parallelization, while do(for)-loops are parallelized with openMP parallelization. For SX6, the computation presented in this paper takes 24,000CPU hours. Figure 4 shows that these openMP and MPI parallelizations have nearly 80 % efficiency in 8-32CPU of SX6.

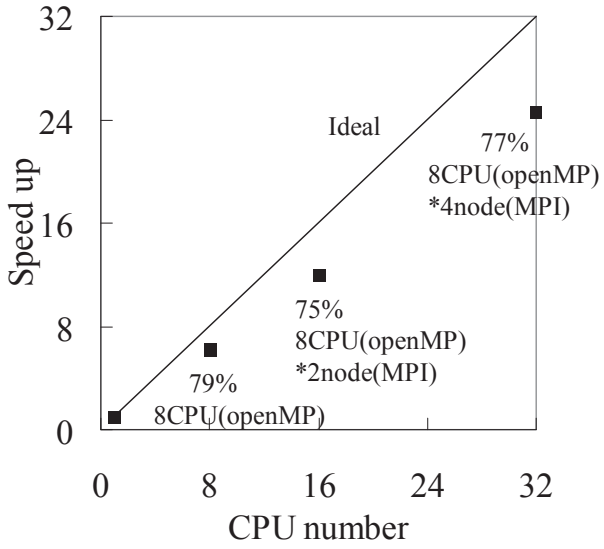


Fig. 4 Parallelization efficiency in SX6 system.

4. Computational Results

Figure 5 shows instantaneous flow-fields and acoustic-fields of the present computation. Acoustic waves are emitted from the various part of the impinging supersonic jet; a upstream region of impingement, an impingement region, and a downstream region of impingement. The overall sound pressure level (OASPL) distribution of near-field shown in Fig. 6 indicates clear acoustic wave generations as discussed above. In this study, the acoustic waves are decomposed into three kinds according to the source position and directivity; (i) Mach waves emitted from shear-layer at the upstream region of impingement, (ii) acoustic waves emitted from the impingement region, and (iii) Mach waves emitted from shear-layer at the downstream region of impingement. The acoustic waves (i) and (iii), seems to be Mach waves because of their directivity, which are observed to be strong towards the downstream region at near-field. On the other hand, the acoustic waves (ii) do not seem Mach wave because it goes upstream

region. These acoustic waves cannot be predicted based on the empirical prediction method, in which the free-jet acoustic wave sources are located along the predicted jet-path.

From the highly resolved simulation with the high speed parallel computers, the phenomenon which is not well-modeled in the empirical prediction method is clarified. This will help us to build a more accurate prediction method.

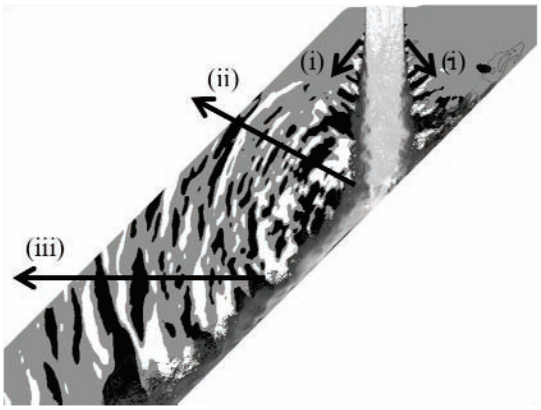


Fig. 5 Instantaneous flow-field and acoustic fields. (dynamic pressure and static pressure)

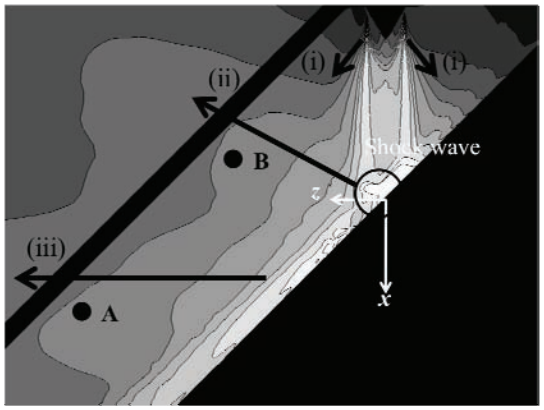


Fig. 6 Overall SPL distributions.

4. CONCLUSIONS

In this presentation, analysis of the supersonic jet impinging to the inclined flat plat is presented. A high-order scheme and high speed parallel computers enable us to conduct highly-resolved simulations. Simulation results clarify the phenomenon, the generation of acoustic waves (ii), which is not well-modeled in the empirical prediction method. This result helps us to build a more reliable prediction method for rocket plume acoustics.

Now, we can compute the rocket plume acoustics at $0.005 < St < 0.5$, whereas the requirement Strouhal number in design is $0.005 < St < 5$. There is 10 times behind between the requirement and the current capable Strouhal number. This 10 times behind in Strouhal number, corresponding to 10,000 times in computational costs, will be covered by the next-generation high speed parallel computers.

ACKNOELEDGEMENTS

This work is partially financed by the Japan Society for the Promotion of Science (Project 19-3209). The first author appreciates their support. Authors are also grateful to Ph. D S. Tsutsumi, for providing the parallelization information of the present computational code.

REFERENCES

1 Eldred, K. M., "Acoustic Loads Generated by Propulsion System," SP 8072, NASA, 1971.
2 Nonomura, T. and Fujii, K., "Computational Analysis of Characteristics and Mach Number Effects on Noise Emission from Ideally Expanded Highly Supersonic Free-Jet," ASME FEDSM 2007-37539, 2007.
3 Nonomura, T. and Fujii, K., "Over-expansion Effects on Mach 3.0 Supersonic Jet Acoustic," AIAA-Paper 2008-2836, 2008.
4 Nonomura, T. and Fujii, K., "Mach Number and Temperature Effects on Mach Wave Emission from Supersonic Jets," AIAA-Paper 2008-6587, 2008.
5 Nonomura, T. and Fujii, K., "Computational Analysis of Characteristics of Mach Wave Sources in Supersonic Free-jet," AIAA-Paper 2009-0016, 2009.
6 Nonomura, T. and Fujii, K., "On Grid and Scheme Resolution for Supersonic Jet Acoustics," Proceedings of WCCM8 and ECCOMAS2008, MS081, 2008.

7 Nonomura, T., Iizuka, N., and Fujii, K., "Increasing Order of Accuracy of Weighted Compact Nonlinear Scheme," AIAA Paper 2007-893, 2007.
8 Shima, E. and Jounouchi, T., "Role of CFD in Aeronautical Engineering (No. 14) -AUSM type Upwind Schemes," Proceedings of the 14th NAL Symposium on Aircraft Computational Aerodynamics SP-34 , January 1997.
9 Deng, X. G. and Zhang, H., "Developing High-order Weighted Compact Nonlinear Schemes," Journal of Computational Physics, Vol. 165, No. 1, 2000, pp. 22–44.
10 Zhang, S., Jiang, S., and Shu, C.-W., "Development of Nonlinear Weighted Compact Schemes with Increasingly Higher Order Accuracy," Journal of Computational Physics, Vol. 227, 2008, pp. 7294–7321.
11 Nonomura, T., Iizuka, N., and Fujii, K., "Uniform Flow Preserving Property of High Order Upwind Scheme on Generalized Coordinate System," Proceedings of International Conference on Computational Fluid Dynamics, 2006.
12 Taylor, E. M., Wua, M., and Martin, M. P., "Optimization of nonlinear error for weighted essentially non-oscillatory methods in direct numerical simulations of compressible turbulence," Journal of Computational Physics, Vol. 223, No. 1, 2007, pp. 384–397.
13 Goto, Y., Nonomura, T and Fujii, K., "Detailed Analysis of Flat Plate Pressure Peaks Created by Supersonic Jet Impingements", AIAA Paper 2009-1289, 2009
14 Boris, J. P., Grinstein, F. F., Oran, E., and Kolbe, R. J., "New Insights into Large Eddy Simulation," Fluid Dynamics Research, Vol. 10, 1992, pp. 199–228.
15 Fujii, K., "Efficiency Improvement of Unified Implicit Relaxation/Time Integration Algorithms," AIAA Journal, Vol. 37, No. 1, 1999, pp. 125–128.

Parallel Simulations of Acoustic Wave Propagation in a 3-D Spherical Model of the Sun

T. Hartlep[⋆], N. N. Mansour[⋆⋆], J. Zhao[⋆] & A. G. Kosovichev[⋆]

[⋆]*Hansen Experimental Physics Laboratory, Stanford University, 452 Lomita Mall, Stanford, CA 94305-4085*
[⋆⋆]*NASA Ames Research Center, Moffett Field, CA 94035-1000*

Abstract: We present a numerical method for simulating the propagation of acoustic waves in a three-dimensional model of the sun. The simulations are used to test and improve helioseismology techniques which provide means for studying the solar interior. Selected results are presented. We also provide some details on the parallel implementation.

Keywords: Acoustics, astrophysics, helioseismology.

1. INTRODUCTION

The dynamics and interior structure of the sun is a challenging and fascinating subject and is actively studied theoretically, numerically, and observationally. Nearly all recent advancements in our understanding of the interior structure and evolution of the sun have been made by studying solar oscillation. Waves are excited by the vigorous convection near the solar surface, travel throughout the complex interior of the sun, and are affected by a spatially varying wave speed and to a lesser extend flows. Oscillations at the solar surface associated with these waves can be observed through ground- and space-based telescopes, such as the *Michelson Doppler Imager* (*MDI*) onboard the *Solar and Heliospheric Observatory* (*SOHO*) spacecraft, and can be used to infer information about the sun's internal structure and composition. This is the science of helioseismology. For a review, see [1]. In general, the inferences are based on simplified models of wave propagation. For instance, the ray approximation is often used. Studying wave propagation in the sun through numerical simulations not only helps to advance our understanding of solar oscillations, but more importantly, the simulations can be used to test and improve our helioseismic inferences.

2. MATHEMATICAL MODEL AND NUMERICAL TECHNIQUE

Simulating the three-dimensional wavefield in a large and complex object as the sun is challenging. Ideally, we would like to simulate the full compressible magneto-hydrodynamic equations to solve

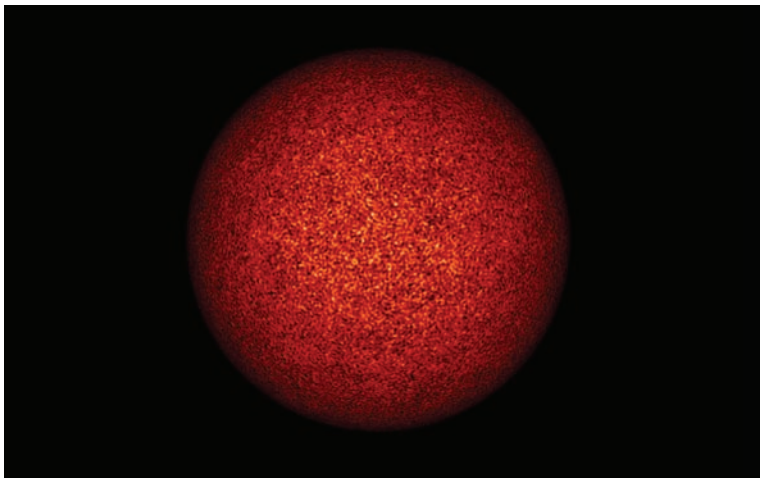


Fig. 1.— *Snapshot of the radial velocity at 300 km above the photosphere of acoustic oscillations in a high-resolution simulation with spherical degrees, l, up to 341.*

for the flows and waves in the sun, but this is unrealistic at the present time and for the foreseeable future due to the numerical cost. Fortunately, the problem of wave propagation can be approximately separated from the flow. Except for the very near surface region, flow velocities in the sun are much smaller than the speed of sound and therefore a perturbation approach is applicable. Acoustic waves can be treated here as small perturbations traveling through a base flow. The base flow itself can be either artificially prescribed or it can be provided by other simulations.

Several approximation are used to derive a trackable set of equations. The oscillations are assumed to be adiabatic, and are driven by randomly forcing density perturbations near the surface. Perturbations of the gravitational potential are neglected. In order to make the linearized equations convectively stable, we also neglect the entropy gradient of the background model. For simplicity, we here show the linearized equations for the case of a background state with zero flow velocities, but these equations account for the more important spatial variation of the wave speed:

$$\partial_t \rho' = -\Phi' + S - \chi\rho', \tag{1}$$

$$\partial_t \Phi' = -\Delta c^2 \rho' + \nabla \cdot \rho' \mathbf{g_0} - \chi\Phi'. \tag{2}$$

Here, ρ' and Φ' are the density perturbations and the divergence of the momentum perturbations associated with the waves, respectively. S is a random function mimicking acoustic sources, c is the background sound speed, and $\mathbf{g_0}$ is the acceleration due to gravity. The damping terms with a coefficient χ, which is zero in the interior and increases above the solar surface, implement an absorbing buffer layer that provides non-reflecting boundary conditions at the upper boundary. For the unperturbed background model of the Sun, we use standard solar model S of [2] matched to a model for the chromosphere [3].

The damping terms are absorbed into the left-hand-side by use of an integrating factor and are therefore treated implicitly. All other right-hand-side terms are treated explicitly. A staggered Yee scheme is used for time integration, and a pseudo-spectral spherical harmonics / B-spline scheme is used for spatial discretization. The variables are expanded in spherical harmonic functions, $Y^{l,m}$, for their angular dependencies (angles θ and ϕ) and fourth order B-splines, B^j, in radial direction (r), e.g.

$$\rho'(r,\theta,\phi,t) = \sum_{m=-m_{max}}^{m_{max}} \sum_{l=|m|}^{l_{max}} \sum_{j=0}^{j_{max}} \hat{\rho}'_{j,l,m}(t) B^j(r) Y^{l,m}(\theta,\phi), \tag{3}$$

with time-dependent coefficients $\hat{\rho}'_{j,l,m}(t)$. All of the operations are performed in this spherical harmonic (l,m) / B-spline (j) coefficient space except for computing the $c_0^2\rho'$-term. This is more efficiently done in (θ,ϕ) / B-spline (j) space using the two-thirds rule for proper dealiasing. The radial resolution of the B-spline method is varied proportionally to the local speed of sound, i.e. the generating knot points are closely spaced near the surface (where the sound speed is small), and are coarsely spaced in the deep interior (where the sound speed is large). Typically, simulations are run with a dealiased l_{max} of 170 or higher and at least 300 B-splines in the radial direction. The fast spherical harmonic transformation used here comes from an existing code developed for simulating convection in spherical shells, see [4,5].

The simulation code has been used on the NASA supercomputers *Columbia* and *Pleiades*, and uses MPI (Message Passing Interface) for inter-processor communication. As mentioned before, most of the computations are performed on data in spherical harmonic (l,m) / B-spline (j) coefficient space. In this case, the data for all l's and j's are in place, and the m's are distributed over the different MPI processes. In (θ,ϕ) / B-spline (j) space, the θ index is distributed, and ϕ and j are in place. Transformations from one storage scheme to the other require MPI messages from each compute process to all other compute processes. The code shows excellent parallel efficiency as long as the number of compute processes is not significantly larger than about half of l_{max}. Beyond this number, the domain decomposition used here results in poor load balancing. Consequently, for a typical l_{max} of 170, we usually use a total of 88 MPI processes of which 87 are compute processes and 1 is used purely for input-output (IO).

3. RESULTS

The main output of these simulations are time series of oscillation velocities at or slightly above the solar surface. These time series are used as artificial data for testing and developing helioseismic inferences. A snapshot of such data is shown in Figure 1. The data is very similar to what is obtained from actual solar observation in which the line-of-sight velocity at some level near the solar surface is measured through doppler effect.

Solar oscillations are trapped inside the sun through reflection at the surface and refraction in the deeper interior, and therefore exhibit resonances. Figure 2 presents the oscillation power spectrum

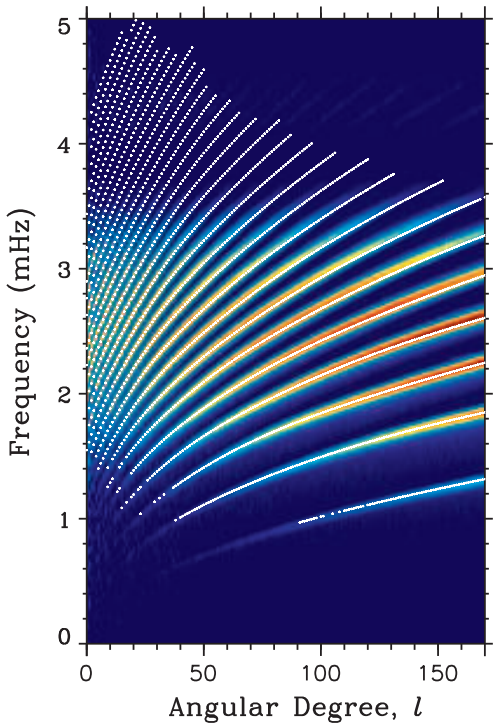


Fig. 2.— *Power spectral density of the radial velocity at 300 km above the photospheric level from a solar acoustics simulations (in color; dark blue denotes zero and red denotes high power density). For comparison, white dots show the frequencies of the resonant acoustic modes measured from actual solar observations.*

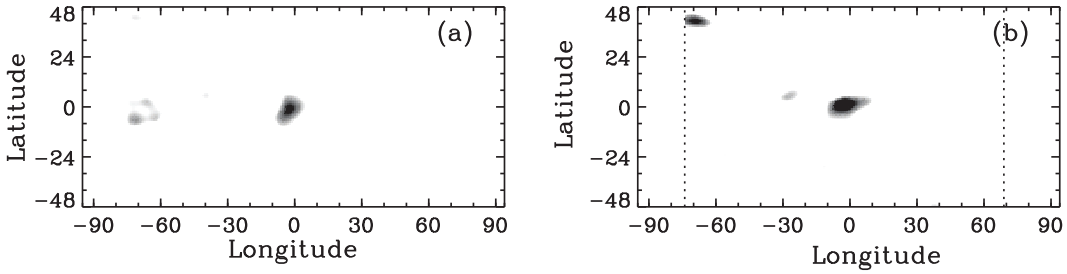


Fig. 3.— *Far-side images computed for a simulation with a single modeled sunspot region at the backside of the sun. (a) and (b) are images derived through two different time-distance helioseismology techniques. Both recover the sunspot region at the center of the solar far-side, but with different amounts of spurious signal.*

as a function of spherical harmonic degree, l, computed for one of the performed simulations. The resonant frequencies of the ridges seen here correspond well with the frequencies obtained from actual solar observations.

One interesting helioseismic technique is so-called far-side imaging which is important for space weather forecasting. Here, one observes the oscillations at the near-side of the sun, the side facing towards earth, and tries to detect sunspot regions on the far-side, the side of the sun facing away from earth and not accessible to direct observations. This is possible because acoustic waves of a certain frequency and wavelength range can travel from the near-side to the far-side and back to the near-side, and because sunspot regions are characterized by sound speed profiles that are different from quiet sun regions. By measuring time-distance correlations of the velocity perturbations on the front-side, one can detect travel time variations of these waves that are associated with regions of modified sound speed on the far-side sun. Figure 3 shows such far-side images computed by two different techniques. In this simulation, a model of a sunspot region has been placed at the far-side of the sun. For more details, see [6]. As in actual observations, only the oscillations on the front-side are used in the far-side imaging methods. Yet, they are able to detect the model sunspot region rather well.

REFERENCES

1. Christensen-Dalsgaard, J. (2002) Helioseismology. *Rev. Mod. Phys.* 74, 1073.

2. Christensen-Dalsgaard, J. et al. (1996) The current state of solar modeling. *Science* 272, 1286.

3. Vernazza, J. E., Avrett, E. H. & Loeser, R. (1981) Structure of the solar chromosphere. III. Models of the EUV brightness components of the quiet sun. *Astrophys. J. Suppl.*, 45, 635.

4. Miesch, M. S. (1998) *Turbulence and convection in stellar and interstellar environments.* PhD thesis. University of Colorado, Boulder.

5. Clune, T. C., Elliott, J. R., Miesch, M. S. & Toomre, J. (1999) Computational aspects of a code to study rotating turbulent convection in spherical shells. *Parallel Comp.* 25, 361.

6. Hartlep, T., Zhao, J., Mansour, N. N. & Kosovichev, A. G. (2008) Validating Time-Distance Far-Side Imaging of Solar Active Regions through Numerical Simulations. *Astrophys. J.* 689, 1373.

Parallel Adaptive Simulation of Weak and Strong Transverse-Wave Structures in H_2-O_2-Ar Detonations

Ralf Deiterding

Oak Ridge National Laboratory, P.O. Box 2008 MS6367, Oak Ridge, TN 37831, USA
(e-mail: deiterdingr@ornl.gov)

Abstract: Two- and three-dimensional simulation results are presented that investigate at great accuracy the temporal evolution of Mach reflection sub-structure patterns intrinsic to gaseous detonation waves. High local resolution is achieved by utilizing a distributed memory parallel shock-capturing finite volume code that employs blockstructured dynamic mesh adaptation. The computational approach and the implemented parallelization strategy are discussed and benchmarked.

Keywords: Supersonic combustion, Mach reflection, structured adaptive mesh refinement.

1. INTRODUCTION

The propagation of detonation waves in gaseous media is a complex multi-scale phenomenon. While gaseous detonations propagate at supersonic velocities between 1500 and 2500 m/s, they inhibit non-neglectable instationary sub-structures in the millimeter range. Transverse pressure waves propagate perpendicular to the detonation front forming triple points with enhanced chemical reaction. The hydrodynamic flow pattern in a triple point is a Mach reflection phenomenon under transient conditions. Depending on the local flow conditions, both double-Mach (aka "strong") and transitional Mach reflection ("weak") structures have been observed in experiments [5].

In the present paper, we discuss results from large-scale parallel simulations of Chapman-Jouguet (CJ) detonations in low-pressure hydrogen-oxygen with high argon dilution. In free space, the triple point movement in such mixtures is very regular leading to a repetitive trajectory pattern of regular "detonation cells". While the detailed hydrodynamic structure of such detonations has been fairly well analyzed by means of numerical simulation for two-dimensional rectangular channels and classified to be of double-Mach reflection type [7, 6], open questions remain for three space dimensions and non-rectangular geometries.

2. COMPUTATIONAL METHOD

The appropriate model for detonation propagation in premixed gases with realistic chemistry is the inviscid Euler equations for multiple thermally perfect species with reactive source terms that read

$$\partial_t \rho_i + \nabla \cdot (\rho_i \vec{u}) = W_i \dot{\omega}_i \,, \quad \partial_t (\rho \vec{u}) + \nabla \cdot (\rho \vec{u} \otimes \vec{u}) + \nabla p = 0 \,, \quad \partial_t (\rho E) + \nabla \cdot ((\rho E + p)\, \vec{u}) = 0 \,, \tag{1}$$

with $i = 1,\ldots,K$. We assume that all K species are ideal gases in thermal equilibrium and that the hydrostatic pressure is given as the sum of the partial pressures $p_i = RT\rho_i/W_i$ with R denoting the universal gas constant and W_i the molecular weight, respectively. The evaluation of the last equation requires the previous calculation of the temperature T. As detailed chemical kinetics necessitate species with temperature-dependent material properties, each evaluation of T involves the approximative solution of an implicit equation by Newton iteration [4]. In here, the chemical production rates are modeled with a hydrogen-oxygen reaction mechanism that considers 34 elementary reactions and the 9 species H, O, OH, H_2, O_2, H_2O, HO_2, H_2O_2 and Ar.

We employ a time-operator splitting approach to decouple hydrodynamic transport and chemical reaction numerically. A semi-implicit Rosenbrock-Wanner method is used to integrate the kinetics within each finite volume cell. Temperature-dependent material properties are derived from look-up tables that are constructed during startup of the computational code. The expensive reaction rate expressions are evaluated by a mechanism-specific Fortran-77 function, which is produced by a source code generator on top of the Chemkin-II library in advance.

Since detonations involve supersonic shock waves, we use a finite volume discretization that achieves proper upwinding in all characteristic fields. The scheme utilizes a quasi-one-dimensional approximate Riemann solver of

Roe-type and is extended to multiple space dimensions via the method of fractional steps. Special corrections are applied to avoid unphysical total densities and internal energies near vacuum due to the Roe linearization, to ensure positive mass fractions, and to prevent the disastrous carbuncle phenomenon. The MUSCL[1]-Hancock variable extrapolation technique is employed to construct a second-order method. The upwind scheme including all modifications is detailed in [4].

In order to consider geometrically complex moving boundaries within an originally Cartesian upwind method, we use some of the finite volume cells as ghost cells to enforce immersed boundary conditions. Their values are set immediately before the original numerical update to model rigid embedded walls. The boundary geometry is mapped onto the Cartesian mesh by employing a scalar level set function that stores the signed distance function. A cell is considered to be an interior cell if the distance in the *midpoint* is positive and is treated as exterior otherwise. The numerical stencil by itself is not modified. Slight approximation errors due to this approach are alleviated by dynamic mesh adaptation. The detailed boundary incorporation for (1) can be found in [3].

The accurate numerical simulation of detonation waves requires a high temporal and spatial resolution particular near the detonation front. In our approach, dynamic mesh adaptation is provided by the structured adaptive mesh refinement (SAMR) algorithm after Berger and Collela [1]. The core idea of the SAMR approach is to cluster cells flagged for refinement with a special algorithm into cache-coherent, rectangular sub-blocks that are enclosed (in here) by two layers of ghost cells. Ghost cells are set to represent physical boundary conditions, by space-time interpolation at level boundaries, and by synchronization with adjacent blocks of the same level. We have implemented the SAMR algorithm in a generic, dimension-independent object-oriented framework in C++. It is called AMROC (Adaptive Mesh Refinement in Object-oriented C++) and is free of charge for scientific use [2]. At present, the SAMR core of AMROC consists of approximately 46,000 lines of code in C++ and approximately 6,000 lines for visualization and data conversion. For computational efficiency, the single-block routines (fluid dynamics and chemical kinetics numerical update, prolongation, restriction) are implemented in Fortran-77.

In the AMROC framework, we follow a rigorous domain decomposition approach, suitable for MPI-based distributed memory parallelization, and partition the SAMR hierarchy from the root level on. The accumulated workload from all levels of the hierarchy is considered to distribute the root level domain at runtime with a space-filling curve algorithm [8]. All higher level domains are required to follow the decomposition of the base level, which can cause the additional splitting of subgrids. The advantage of the rigorous domain decomposition approach is that it is comparably easy to implement, with hierarchy recomposition and subgrid synchronization being the only parallel operations, and overall work is well balanced. However, the work on each levels is not perfectly distributed, which causes slight delays during subgrid synchronization (in operation *Boundary setting*). We have found the approach well suitable for parallel three-dimensional computations for up to a few hundreds of CPUs. Results from a two-dimensional scalability test are depicted in Fig. 1. The simulation approximates the shock-induced combustion around a sphere that travels at supersonic speed through a hydrogen-oxygen-argon mixture [3]. The computation is carried out in the frame of reference of the body, leading to a steady flow field and mesh refinement, and uses a base

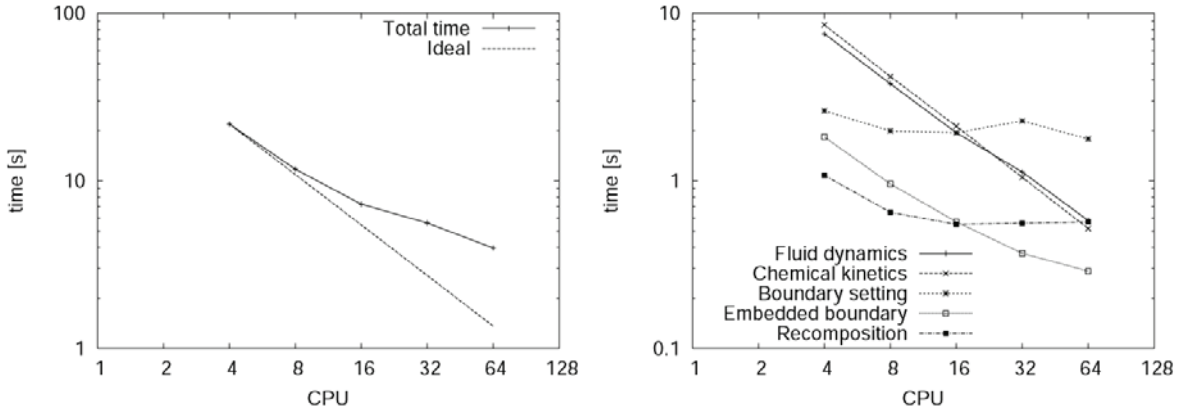


Fig. 1: *Strong scalability test for the two-dimensional chemically reactive SAMR code. Total time required for one full integration and refinement cycle (left) and for the most important operations (right).*

[1] Monotone Upstream-centered Schemes for Conservation Laws

Fig. 2: *Schlieren plot of density on triple point tracks (left) and on refinement regions (shaded gray, middle and right, successive enlargement into two triple points) for $\varphi = 45^{o}$ and t = 150 µs simulated time.*

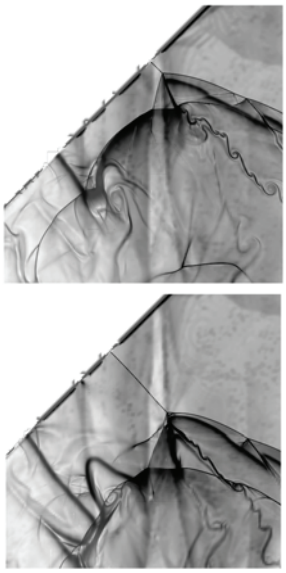


Fig. 3: *$\varphi = 45^{o}$: transverse detonation wave (t=200, 210 µs).*

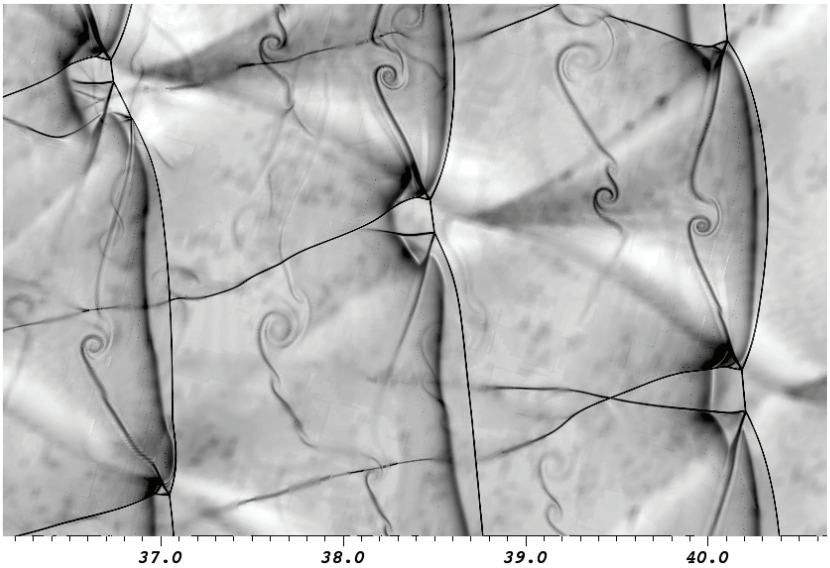


Fig. 4: *$\varphi = 15^{o}$: formation of new transverse wave and strengthening of the corresponding triple point (t = 200 µs, 210 µs, and 220 µs simulated time).*

mesh of only 70 x 40 cells and 3 additional levels refined by factor 2. The test was run on a cluster of Intel Xeon 3.4 GHz dual-processors connected with a Gigabyte Ethernet network. As can be inferred from the right graphic of Fig. 1, the numerical single-block operations for fluid dynamical and chemical kinetics update scale linearly, however the expense of the communication-dependent operations for synchronization and hierarchy recomposition remains basically constant for larger CPU counts.

3. RESULTS

First, we study two-dimensional CJ detonations in $H_2 : O_2 : Ar$ mixtures of molar ratios 2 : 1 : 7 at initial temperature 298 K and pressure 10.0 kPa that propagate through smooth elbow pipes of the bending angles $\varphi = 15^{o}$, 30^{o}, 45^{o}, and 60^{o}. One-dimensional detonation structure theory after Zel'dovich, von Neumann, and Döring (ZND) predicts a detonation velocity of 1638.5 m/s, a von Neumann Pressure of ~270 kPa, and a reaction length of $l_{ig} \approx 0.878$ mm for this mixture.

In a preparational computation (not shown), the one-dimensional ZND solution is placed on a rectangular two-dimensional mesh and perturbed, leading to a periodic cellular oscillation with cell width ~1.6 cm. A snapshot of a single cell is then reproduced periodically and used to initialize the flow field with the detonation front

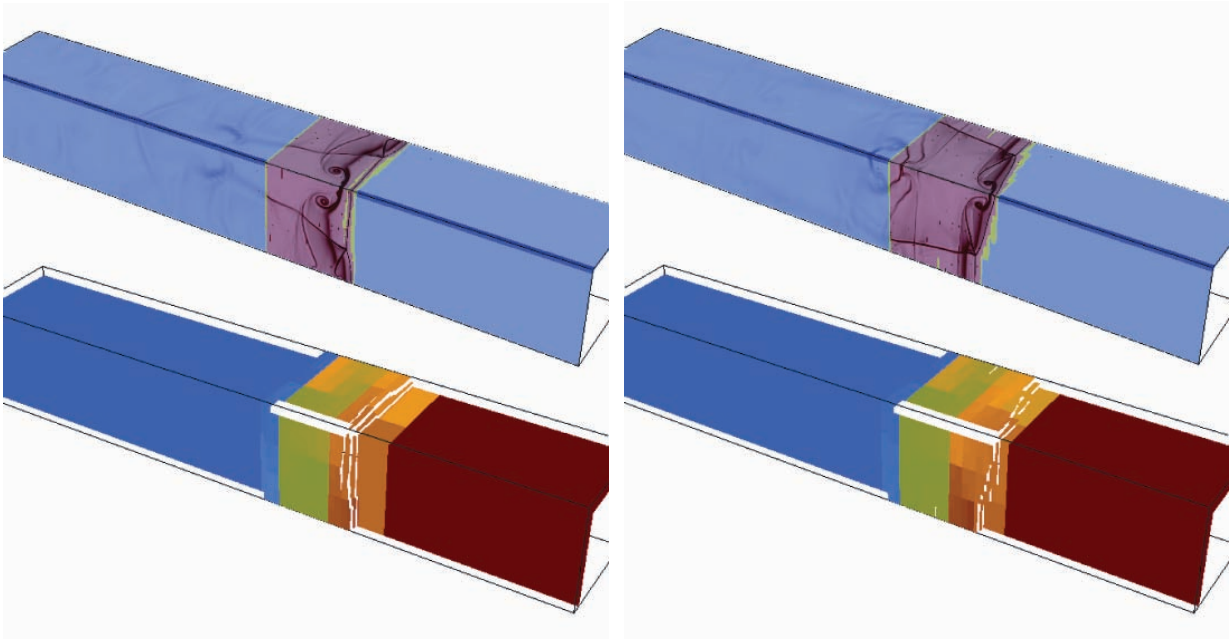


Fig. 5: *Schlieren plot of the density on refinement levels and domain distribution to 128 CPU (indicated by color) in the first (left) and second (right) half of a detonation cell.*

approximately 13 cm before the beginning of the curved section. By using a pipe of only 8 cm width a strong influence of the geometry on the detonation front structure can be expected.

To accommodate a reduction of the induction length when the detonation wave gets compressed, all computations use an effective resolution of 67.6 Pts/l_{ig}, which is achieved by four additional levels of Cartesian mesh adaptation with refinement factors 2, 2, 2, and 4. A physically motivated combination of scaled gradients of pressure, density and heuristically estimated relative errors in the mass fractions is applied as adaptation criteria [3]. For instance, for bend angle $\varphi = 60^o$, with a base mesh of 1200 x 992 cells, the adaptive computation uses approximately 7.1 M to 3.4 M cells on all and 4.8 M to 1.8 M cells on the highest level instead of ~1,219 M in the uniform case. The calculations were run on 64 Intel Xeon 2.4 GHz-dual-processor nodes with Quadrics interconnect and required nevertheless ~70,000 h CPU each (~23 days wall time). The extraordinary high efficiency in capturing only the essential features near the detonation front is illustrated in Fig. 2, in which a double-Mach reflection pattern is clearly resolved. Note that the reaction zone appears in Fig. 2 as the diffused black line following the sharp shock wave at the head of the detonation front.

As the detonation propagates through the bends, it gets compressed and consequently over-driven near the outer wall, but a shock wave diffraction occurs near the inner wall. For $\varphi = 15^o$, detonation wave expansion and compression in the bend only lead to a temporary change in detonation cell size. From $\varphi = 30^o$ on, an increasing decoupling of leading shock and reaction zone occurs as the shock wave diffraction causes a temperature decrease below the limit of detonability. Isolated unreacted pockets can also be observed, cf. Fig. 2. While the decoupling is only partial for $\varphi = 30^o$, a temporary detonation failure clearly occurs for $\varphi \geqq 45^o$. A transverse reignition wave arises from the successfully transmitted region and reinitiates the detonation in the decoupled area, cf. Fig. 3. The triple point intersection between transverse detonation and incident shock forms a double-Mach reflection pattern of exceptional strength. After leaving the bend, and possibly reigniation, the triple point oscillation at the head of the detonation front returns to the original periodicity. An example for the formation and strengthening of new transverse waves is depicted for three output time steps in Fig. 4. The triple point traveling from the lower left to the upper right corner is initially formed as a weak structure of transitional Mach reflection type; through the triple point collision in the center of the graphic it strengthens into a double-Mach reflection, which can be clearly inferred from the characteristic curvature of the subsequent triple point track.

As a three-dimensional example, we discuss a simulation to analyze the detailed triple point structure for a CJ detonation in $H_2 : O_2 : Ar$ / 2 : 1 : 7 at initially 298 K and pressure 6.67 kPa. The detonation cell width in free space is

then $\lambda \approx 3.0$ cm [7, 6] and the ZND induction length $l_{ig} \approx 1.4$ mm. In rectangular three-dimensional domains, the triple points manifest themselves as orthogonal triple point lines [9]. A detailed hydrodynamic analysis uncovers that, although the detonation velocity is unaltered, the fluctuations in pressure, temperature, and therefore induction length are considerably larger in 3d than in 2d. The discussed simulation is carried out in a frame of reference attached to the detonation front. The domain has the dimensions [0 cm, 10 cm] x [0 cm, 1.5 cm] x [0 cm, 1.5 cm] to simulate exactly 1/4 of a regular detonation cell. A constant inflow with 1626.9 m/s is applied at the right, outflow conditions at the left boundary. Symmetry boundary conditions are used at all other sides.

The computation uses a base mesh of 400 x 24 x 24 and two additional levels of mesh adaptation with refinement factors 2, 4 giving an effective resolution of 44.8 Pts/ l_{ig}. Refinement criteria are chosen similarly as before, where all refinement flags are overall deleted in the range $0\,\text{cm} < x < 4\,\text{cm} + v_0 t$ with $v_0 := 20$ m/s. After a simulation time of ~600 μs a regular cellular oscillation with identical strength in y-and z-direction can be observed, cf. Fig. 5. The fine effective resolution achieved allowed the unambiguous classification of the hydrodynamic flow pattern at triple point lines as the weak structure of a transitional Mach reflection, which is quite remarkable since the same mixture exhibits the strong structure of double-Mach reflection type in the two-dimensional case [6, 4] (see also Fig. 2). This computation was run on 32 nodes of a Compaq AlphaServer quad-core system with high-speed Quadrics interconnect at Los Alamos National Laboratories and required ~51,000 h CPU, which corresponds to ~16.6 days wall time. A breakdown of the compute time is depicted in Table 1. The adaptive computation uses approximately 16.5 M cells on average instead of 118 M in the uniform case, cf. upper row of Fig. 5. The lower row of Fig. 5 visualizes by different color the domain decompositions of the evolving hierarchy to 128 CPU with the refinement levels elevated. These pictures illustrate the good stability of the partitioning methodology for small changes in the workload.

Table 1: Breakdown of the compute time for the 3d simulation.

Task	%
Fluid dynamics	37.6
Chemical kinetics	25.1
Boundary setting	24.4
Recomposition	6.6
Misc.	6.3
Total [h CPU]	**~51,000**

ACKNOWLEDGEMENTS

This work is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy (DOE) and was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725. The presented computations have been carried out while the author was at the California Institute of Technology and was supported by the ASC program of the Department of Energy under subcontract No. B341492 of DOE contract W-7405-ENG-48.

REFERENCES

1. Berger, M. and Colella, P. (1988). Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82:64–84.
2. Deiterding, R. (2002). AMROC - Blockstructured Adaptive Mesh Refinement in Object-oriented C++. Available at http://amroc.sourceforge.net.
3. Deiterding, R. (2009). A parallel adaptive method for simulating shock-induced combustion with detailed chemical kinetics in complex domains. *Computers & Structures*, in press.
4. Deiterding, R. and Bader, G. (2005). High-resolution simulation of detonations with detailed chemistry. In G. Warnecke (ed.), *Analysis and Numerics for Conservation Laws*, pages 69–91, Springer.
5. Fickett, W. and Davis, W. C. (1979). *Detonation*. University of California Press, Berkeley and Los Angeles.
6. Hu, X. Y., Khoo, B. C., Zhang, D. L., and Jiang, Z. L. (2004). The cellular structure of a two-dimensional H2/O2/Ar detonation wave. *Combustion Theory and Modelling*, 8:339–359.
7. Oran, E. S., Weber, J. W., Stefaniw, E. I., Lefebvre, M. H., and Anderson, J. D. (1998). A numerical study of a two-dimensional H2-O2-Ar detonation using a detailed chemical reaction model. *Combust. Flame*, 113:147–163.
8. Parashar, M. and Browne, J. C. (1996). On partitioning dynamic adaptive grid hierarchies. In *Proc. of the 29th Annual Hawaii Int. Conf. on System Sciences*.
9. Williams, D. N., Bauwens, L., and Oran, E. S. (1997). Detailed structure and propagation of three-dimensional detonations. In *Proc. of the Combustion Institute*, pages 2991–2998.

A Study on Combustion Flow Dynamics by High-Fidelity Numerical Simulation

Junji SHINJO*, Shingo MATSUYAMA*, Yasuhiro MIZOBUCHI*, Naoyuki FUJITA*, Ryoji TAKAKI, Yuichi MATSUO***

**Aerospace Research and Development Directorate, Japan Aerospace Exploration Agency (JAXA), Chofu, Tokyo, 182-8522 JAPAN*
(Tel: 81-422-40-3316; e-mail: {shinjo.junji, matsuyama.shingo, mizobuchi.yasuhiro, fujita.naoyuki, matsuo.yuichi}@jaxa.jp)
*** Institute of Space and Astronautical Science, JAXA, Sagamihara, Kanagawa, 229-8510, JAPAN*
(e-mail: takaki.ryoji@jaxa.jp)

Abstract: To predict various complicated phenomena in rocket or aero engine combustors such as turbulence-flame interaction and fuel atomization, we have developed high-fidelity numerical simulation methods based on parallel Large-Eddy Simulation (LES) or Direct Numerical Simulation (DNS). The simulations are conducted on the JSS1, which is a new large computer system installed at JAXA at FY2008. In this paper, the system configuration and features of JSS1 are described and several combustion-related simulation results are presented.

Keywords: Combustion, LES, DNS, Engine combustor, High-fidelity simulation

1. INTRODUCTION

In aerospace engine development for both rocket engines and jet engines, one of the critical components is a combustor. However, phenomena occurring inside the combustor are highly complicated and difficult to observe in experiments. Numerical simulations to understand combustion-related phenomena such as flame stabilization, flame-turbulence interaction and liquid fuel atomization are becoming possible using high-fidelity numerical simulations like parallel Large-Eddy Simulation (LES) or Direct Numerical Simulation (DNS) techniques. In this paper, two examples of combustion-related flow field are shown to demonstrate the effectiveness of large-scale parallel computation. The new computer system at JAXA is used.

2. SIMULATION MODEL AND PARALLEL IMPLEMENTATION

The first simulation deals with hydrogen non-premixed combustion at a supercritical pressure in a rocket combustor. LES is used to obtain unsteady solutions. The Soave-Redlich-Kwong equation of state is used to include high-pressure and low-temperature physical properties [1]. Mixture fraction represents the mixing and combustion and the flame is modeled by 1-D flamelet assumption. The numerical scheme is based on AUSM+up and the Smagorinsky subgrid model is incorporated. The second simulation solves liquid fuel atomization, which is a key process in subcritical liquid-fueled engines. A two-phase flow solver [2] is utilized in a direct simulation context. The 3-D incompressible Navier-Stokes equations with surface tension are solved. The surface tension is evaluated by the Continuum Surface Force (CSF) method. Pressure is determined by solving the pressure Poisson equation. The numerical scheme for solving flow advection is based on the Cubic Interpolated Pseudo-particle (CIP) method. The phase interface is captured by the Level Set method and a modified version of Volume-of-Fluid (VOF) method, called Multi-interface Advection and Reconstruction Solver (MARS) method, is combined to assure volume conservation. For parallel implementation between processes, MPI is applied, whereas on a node where one process is allocated with a multicore chip, automatic parallelization based on thread parallelism is applied.

3. JSS1

We did these simulations on *JSS1: JAXA Supercomputer System 1*. The system is described below.

3.1 System Configuration

JSS1 consists of Compute Engine part that has totally 141TFLOPS peak performance, Storage part that has 11PB disk and tape devices, and Distributed Environment Consolidation part that realize synthesized user environment (Fig. 1). Compute Engine part has four types of system, namely M-, P-, A-, and V-System. Table 1 shows this classification. Storage part consists of 1PB RAID5 disk system that has 180 FC link and 25GB/s actual I/O performance and 10PB LTO4 libraries that has 40 LTO4 drives and 8 LTO3 drives. Backbone of Distributed Environment Consolidation part is incarnated on SINET3 [3] by using Virtual Private Network (VPN) technology. The L-systems, which supply login, compile and debug environment to remote site's users, and the J-SPACE, which is distributed remote file system, are connected to the backbone.

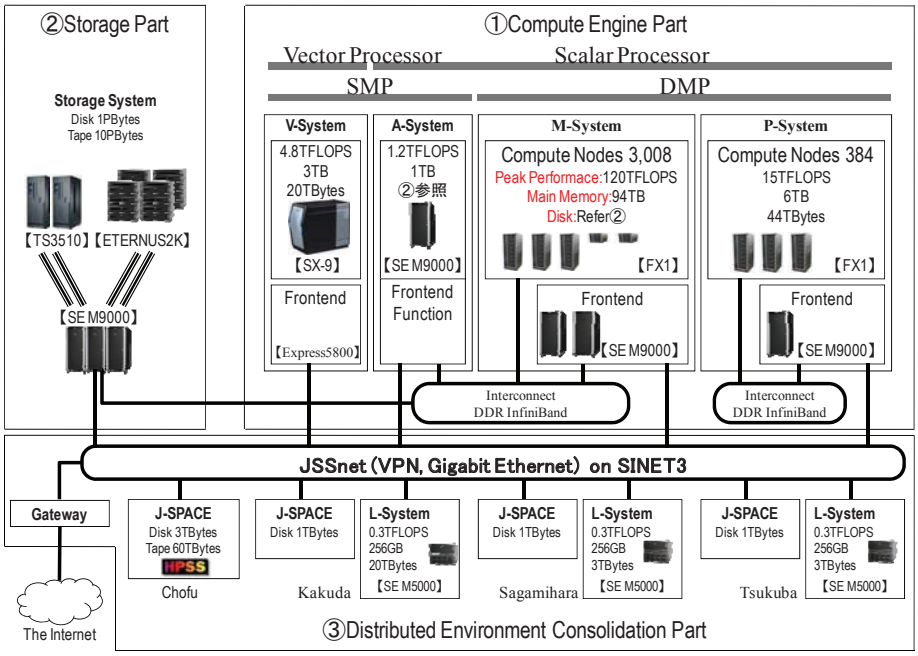


Fig. 1: *JSS configuration*

Table 1: Compute Engine classification

System name	M-System	P-System	A-System	V-System
Processor type	Scalar	Scalar	Scalar	Vector
Processor – memory connection type	Distributed	Distributed	Shared	Shared
Usage	Unclassified	Classified	Unclassified	Unclassified
Number of nodes	3,008	384	1	3
Number of CPUs	3,008	384	32	48
Number of cores	12,032	1,536	128	48
Peak TFLOPS	120	15	1.2	4.8
Total main memory	94TB	6TB	1TB	3TB
Memory per node	32GB	16GB	1TB	1TB
Brand	Fujitsu FX1	Fujitsu FX1	Fujitsu Sparc Enterprise M9000	NEC SX-9

3.2 Feature of Compute Engine

M-System is the largest resource in Compute Engine. The system has some features on efficiency for large-scale numerical simulations.

3.2.1 Multi-core Programming Model

About FORTRAN's nested DO-Loop, on conventional scalar parallel computers, the outermost DO-loop should be parallelized manually, while M-System parallelizes the innermost DO-loop automatically by the compiler (Fig. 2). This feature advocates a new programming model. Figure 3 shows the comparison between the traditional and new programming models. Not only M-System but also newly coming parallel computers will have multi-core CPUs. As shown in Fig. 3(b), if we use the traditional programming model, each process should be assigned on each core manually. On the other hand, the new programming model assigns multiple threads on one node automatically. This means that a user does not need to awake to use multi-core CPUs. In addition, well accumulated vectorization techniques can be applied to automatic parallelization of the innermost DO-loop.

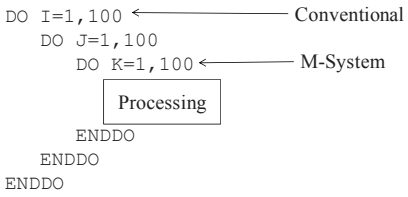


Fig .2: *Parallelization of DO loop*

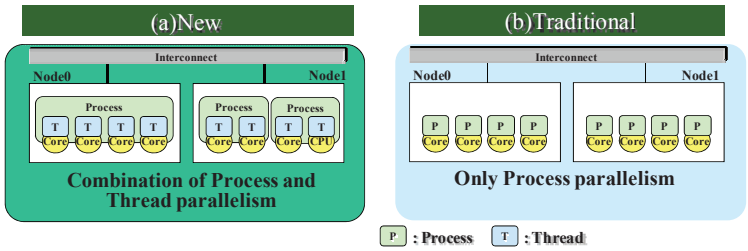


Fig .3: *JSS M-System programming model*

3.2.2 Intra and Inter Node Hardware Barrier Mechanism

M-System has 3,008 compute nodes and 12,032 cores. Each node contains a quad-core 2.5 GHz SPARC64 VII processor and 32GB of memory. As the number of processing elements increases, synchronization overhead time for user program and OS process increases. Especially on M-System, because inner DO-loops are automatically parallelized, the number of synchronization increases. So speed-up of synchronization should be needed. So M-System has hardware barrier mechanisms on Intra-node and Inter-node. Figure 4 shows a difference of speed between hardware and software barrier mechanisms. From these advantages, M-System has world top class sustained performance 91.19% on LINPACK.

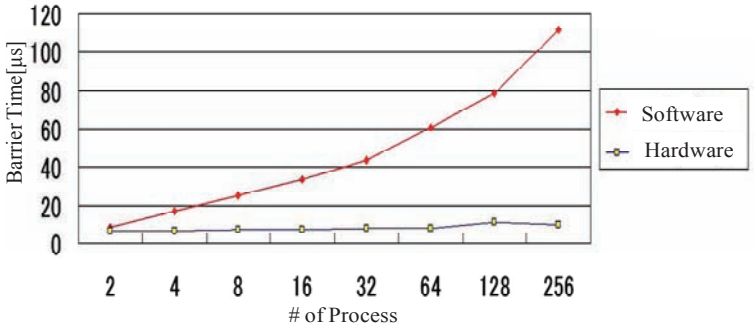


Fig .4: *Soft barrier vs. Hard barrier*

3.2.3 Parallel Execution Performance

In this section, we show the efficiency of parallel execution performance on large numbers of cores. Table 2 shows some parallel applications which were executed on M-System. From P1 to P5 are real application code in aerospace, and P6 is Linpack benchmark program.

Table 2: Parallel applications

Code	Application field	Numerical Method
P1	Combustion	FDM+Chemistry
P2	Aeronautics	FVM(Structured)
P3	Turbulence	FDM+FFT
P4	Space Plasma	PIC
P5	Aeronautics	FVM(unstructured)
P6	Linpack	(High Performance Linpack)

Table 3 shows the result of efficiency. This efficiency measured scale-up-problem-size case, not a fixed-problem-size case. Efficiency e is defined as follows;

$$e = \frac{\left(a1 * \frac{b2}{a2}\right) \Big/ \frac{b3}{a3}}{b1} = \frac{a1 * b2 * a3}{b1 * a2 * b3} \qquad (1)$$

Here, $b2/a2$ is scale-up of problem size, $b3/a3$ is scale-up of computing resource, namely core resource. As one can see in Table 3, the efficiency is from 62% to 91% in using over 2,000 cores in real aerospace applications. Measuring in Linpack program, the efficiency comes to 97%. That means M-System has good parallel performance on large numbers of cores.

Table 3: Result of parallel execution performance

Code	Execution on single node			Execution on multi node			Efficiency
	Exec time [s] a1	# of grids (# of floating point operations) a2	# of cores a3	Exec time [s] b1	# of grids (# of floating point operations) b2	# of cores b3	e
P1	131.0	1,728,000	4	143.3	1,285,632,000	2,976	0.914
P2	71.0	512,000	4	91.5	384,000,000	3,000	0.776
P3	346.8	1,572,864	4	491.7	805,306,368	2,048	0.705
P4	164.0	65,536	4	193.0	49,152,000	3,000	0.850
P5	142.0	4,173	4	181.6	2,492,921	3,000	0.622
P6	3566.4	$(1.3361*10^{14})$	4	218376.38	$(2.4101*10^{19})$	12,032	0.979

4. RESULTS AND DISCUSSION

Figure 5 shows an LES result in a model LOX/GH_2 (liquid oxygen and gaseous hydrogen) rocket engine combustor. The simulation was conducted with 11 million grid points and 36 parallel regions. Very fine vortex structures around the nozzle are well captured and the flame spreading angle is in good agreement with the experimental data [1]. By using LES, unsteady flame dynamics can be understood.

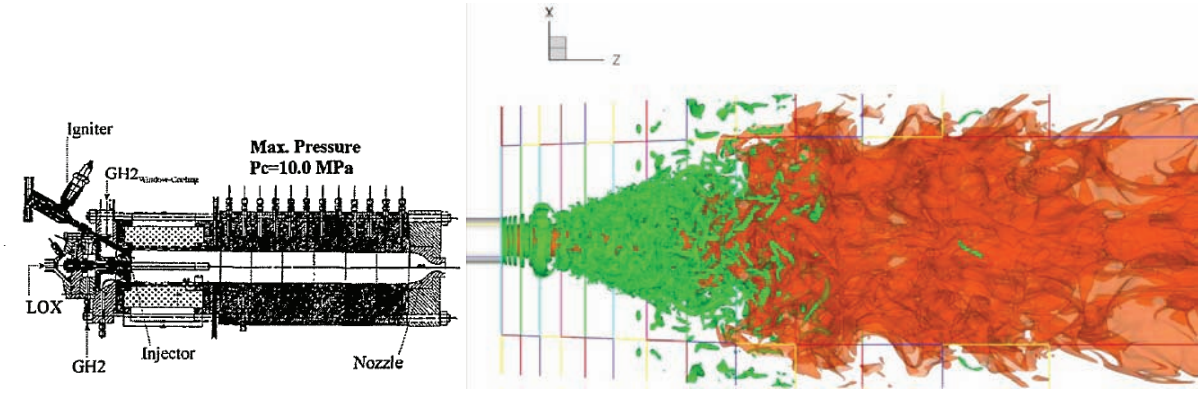


Fig. 5: *Supercritical cryogenic model rocket combustor [4] and the present simulated result. The fine structures in green are vortices and the red iso-surface represents T=1000K.*

Figure 6 shows a direct simulation result of liquid fuel jet injection under 30atm ambient pressure. The number of grid points is 110 million and the computational domain is divided into 64 parallel regions. The injection nozzle diameter is 0.1mm and the injection velocity is 100m/s. Surface instability after injection is well captured and subsequent formation of ligaments (thread-like structures) and droplets can be observed. This kind of information is difficult to obtain in experiments and it is expected to improve our understanding of atomization mechanism and modeling. A larger simulation with 6 billion grid points and 1,440 parallel regions is being conducted now to examine smaller scale phenomena.

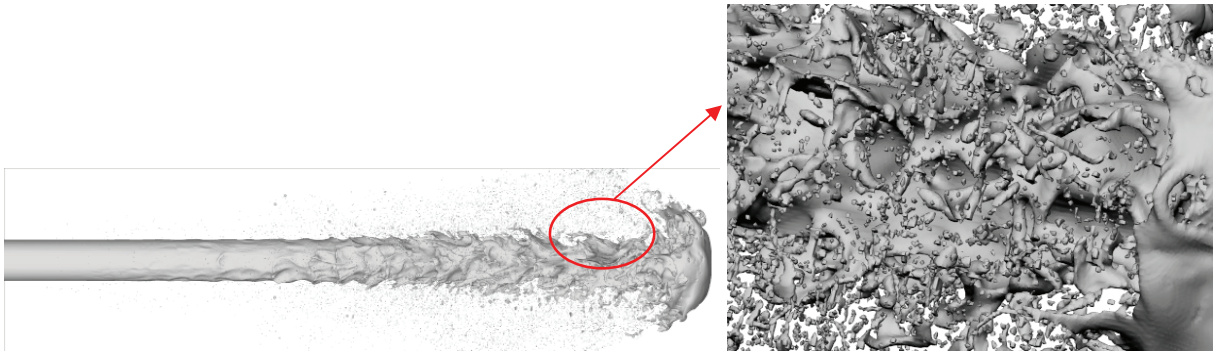


Fig. 6: *Liquid fuel injection simulation*

5. CONCLUSION

In this paper, the new supercomputer system JSS1 installed at JAXA and simulation results of combustion-related flows are briefly introduced. These kinds of complex flow computations are becoming possible with recent faster parallel computer performance. For practical engine development applications, understanding complicated combustion flow fields by numerical simulation is becoming more and more significant.

REFERENCES

1. Matsuyama, S., Shinjo, J., Ogawa, S., Mizobuchi, Y. (2008) "LES of supercritical-pressure LOX/GH2 coaxial jet flame by flamelet model", *Proceedings of the 46th Japanese Symposium on Combustion*, Kyoto
2. Shinjo, J., Matsuyama, S., Mizobuchi, Y., Ogawa, S., Umemura, A. (2009) A numerical study on ligament disintegration mechanism by propagative capillary waves, *Atomization*, in press
3. National Institute of Informatics, "SINET3", http://www.sinet.ad.jp/?set_language=en
4. Ivancic, B., Mayer, W., Brueggemann, D., Kruelle, G. (1999) "Experimental and numerical investigation of time and length scales in LOX/GH2-rocket combustors" *AIAA Paper* 99-2211

UNSTRUCTURED/OVERSET GRID METHODS

An Overset Unstructured Grid Method for Parallel Solvers

H.U. Akay, R. U. Payli, J. Liu, and A. Ecer

Computational Fluid Dynamics Laboratory
Department of Mechanical Engineering
Indiana University-Purdue University Indianapolis
723 W. Michigan St., Indianapolis, IN 46202 – USA

Abstract: A parallel overset grid method is developed for solution of moving body CFD problems using unstructured grids. The developed overset parallel library is suitable for unsteady solution of complex flow fields and complex motions. It can easily be implemented to existing flow solvers with minimum code modifications. The efficiency of the developed library is studied for different number of processers.

Keywords: Overset Grid, Hole Cutting, Donor Cell Searching, Grid Partitioning, and Parallel CFD.

1. INTRODUCTION

The overset grid method is used as a method in computational fluid dynamics to simplify the grid generation of complex geometries or to enable relative motion between different components. This method utilizes a set of grids to discretize the domain with each component of the grid generated locally around a portion of the geometry. The grids are allowed to overlap without any point-to-point match requirement to each other. This ability greatly simplifies the grid generation process for complex geometries.

The most implementations of the overset method are based on structured grids [1, 2, 3]. The difficulty of generating structured grids around complex geometries is the main disadvantage of this approach. The most critical issue of the structured overset approach has been that the required number of grid points to be overlapped increases fast as the complexity of the geometry increases. During the recent years, unstructured grid flow solvers have been gaining popularity because of their flexibility to handle complex geometries. The unstructured overset grid methods also offer additional flexibility to flow problems with multi-component motions.

The implementation of the overset grid techniques into existing structured and unstructured flow solvers typically requires extensive and tedious code modifications. The modifications required include adding hole cutting, surface grid assembly, near-body grid assembly, and off-body grid assembly. Because most of the overset functions are straightforward and essentially duplicated among flow solvers to facilitate rapid implementation of the overset grid capability into the flow solvers, two popular libraries: Donor Interpolation Receptor Transaction Library (DIRTLIB) [4] and the Structured, Unstructured, and Generalized Grid Assembler (SUGGAR) [5] have been developed in the literature. While these libraries are very useful, there are restrictions in using them for multi-national projects, since they are subjected to export control laws. In this paper, we will present an in-house parallel library developed in parallelization of our unstructured overset grid method, OverSun [6], which can be easily coupled with other parallel flow solvers. Applications of the library will be demonstrated for moving-body problems in the full paper.

The program OverSun has following advantages over others:

- The hole cutting process is done in a systematic and automatic fashion
- An efficient search algorithm is used

- Instead of using wall boundaries for hole cutting, one cell overlap is used to reduce the computation time
- Integrating the solvers requires a few changes instead of the extensive function calls needed with DIRTLIB and SUGGAR
- Modeling of complex geometries and movements are easy
- Once a grid around a moving body (minor grid) is generated, it is coupled with the background grid (major grid) for forming a single grid for the entire flow domain and the arbitrary movements of the body surrounded by the minor grid
- Handling of the extreme condition of the wall-to-wall contact cases are done automatically, when the wall boundaries of the moving body just touch a fixed wall, blocking the flow passage

2. OVERSUN PARALLEL LIBRARY

The OverSun has overset grid and domain partitioning modules, which are coupled with each other via Message Passing Library (MPI). Figure 1 shows flow of the OverSun information from combination of the major (background) and the minor (moving) grids of the overset system to the partitioning of the combined grid for parallel computing.

2.1 OverSun Grid Module

The overset grid module is capable of automatically cutting hole(s), efficiently searching donor cells, reducing computational time via one-cell overlapping. Figures 2 and 3 show grids before and after hole cutting steps for external and internal flow problems, respectively (details are in [6]).

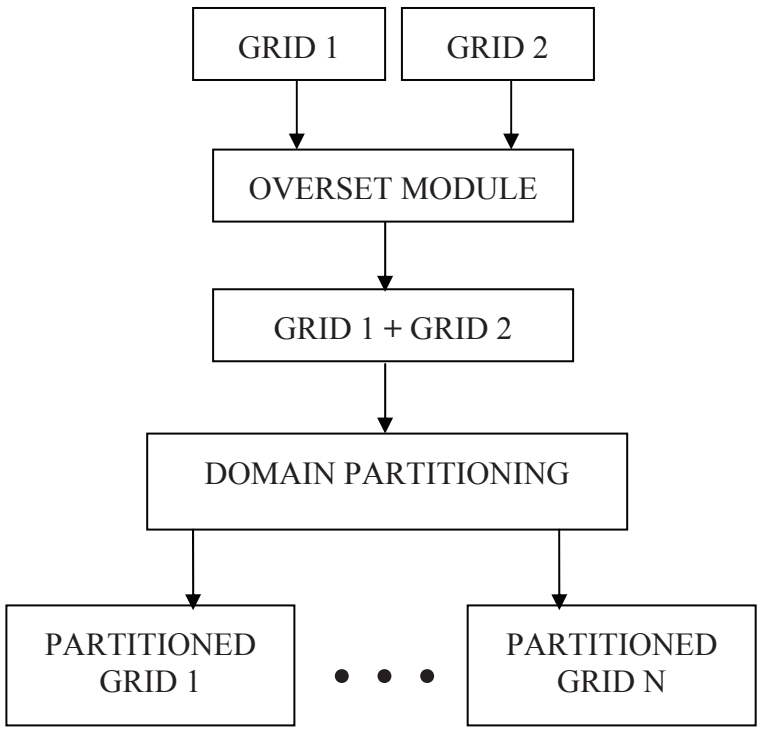


Fig. 1: *Flow of the Parallel OverSun information from combining of the minor and major grids (Grid 1 and Grid 2) to further partitioning of the combined grid for parallel computing.*

2.2 Domain Partitioning Module

This module is responsible for partitioning the grid generated by the overset grid module for Single Program Multiple Data (SPMD) parallel solver(s). The grid partitioning steps and features are as follows:

- An initial partition is generated using the Metis [7] graph partitioning library. The Metis converts the grid into a graph and then partitions it into desired number of partitions using graph partitioning
- Each processor carries the second level of partitioning using the initial partition information. The one element-overlapped interfaces are constructed in second partitioning phase
- After these phases, the files contain necessary information such as partition coordinates, connectivity, boundary conditions, node and element interface information written by each processor
- For transient problems with moving bodies, these are repeated for at each time step. Hence the efficiency of the parallel algorithm developed becomes important

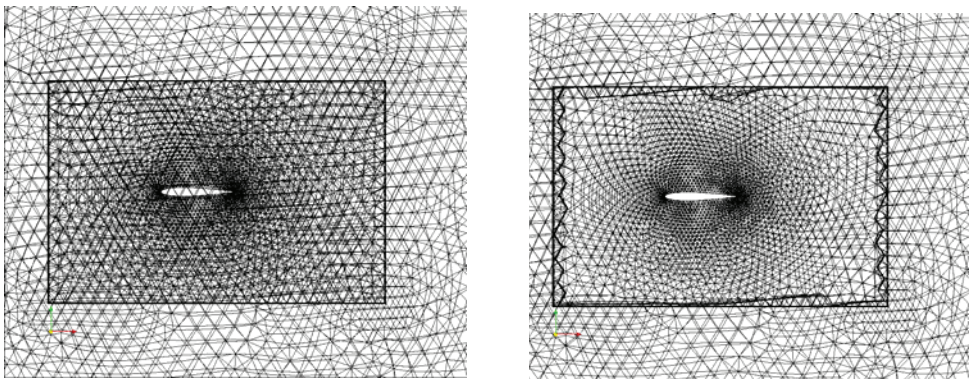


Fig. 2: *Background and minor grids before and after hole cutting operations – a typical external flow scenario.*

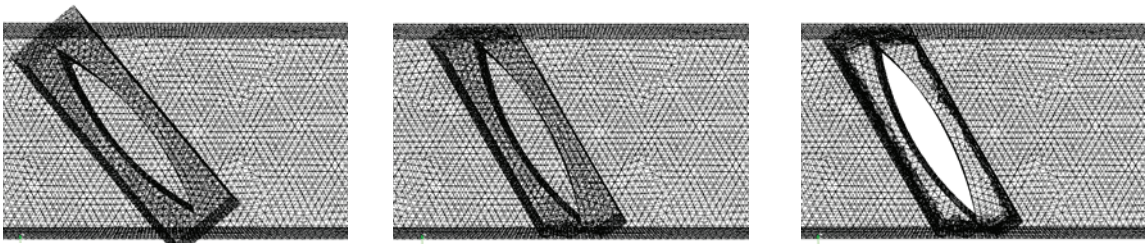


Fig. 3: *Background and minor grids before and after hole cutting operations – a typical internal flow scenario (note that the minor/moving grid intersects with the fixed wall boundaries).*

The partitioned domain for parallel computing of the problem of the grid in Figure 3 is shown in Figure 4, where different partitions are indicated with different colors. In this problem the moving part is a three-dimensional valve placed in a channel at the near-complete blocking stage.

Fig. 4: *Partitioned domain for parallel computing.*

2.3 Coupling of OverSun with Flow Solvers

The code coupling is a technique which allows the data exchange between two or more codes in a loosely coupled fashion. The OverSun can be loosely coupled with the flow solvers via MPI. A typical MPI application has a general communicator to allow application processors communicate with each other for data exchange. When the OverSun is coupled with a flow solver, this communicator is divided into two communicators, one for the OverSun, the other for the flow solver. This way the OverSun processors communicate with each other via OverSun's communicator while the flow solver processors communicate with each other via the solver's communicator. When the data exchange is needed between the OverSun and the flow solver, the general communicator is used.

The coupled OverSun and the flow solver work as follows: When OverSun generates the composite grid data such as grid coordinates, it sends the connectivity and interpolation data to the flow solver via the general communicator. The solver uses these data to solve the problem for the whole flow domain. If the body movement exists while the solver working on the data, the OverSun generates a new set of data for the next time step.

3. RESULT TO BE PRESENTED

In the full paper, applications of the developed parallel overset code for solution of moving-body problems with different number of processors and flow conditions will be demonstrated. Parallel efficiency of the system will be investigated.

REFERENCES

1. Suhs, N.E. and Tramel, R.W. (1991). *PEGSUS 4.0 User's Manual.* AEDC-TR-91-8, Arnold Engineering Development Center, Arnold AFB, TN.
2. Henshaw, W.D. (2002). *The Overture: Overlapping Grid Generator.* Technical Report UCRL-MA-132237, Lawrence Livermore National Laboratory.
3. Buning, P.G. (1993). *OVERFLOW User's Manual.* Version 1.6ad, NASA Ames Research Center, CA.
4. Noack, R.W. (2005). DIRTLIB: A Library to Add an Overset Capability to Your Flow Solver. *AIAA Paper 2005-5115, 17th AIAA Computational Fluid Dynamics Conference*, Toronto, Ontario, Canada.
5. Noack, R.W. (2005). SUGGAR: A General Capability for Moving Body Overset Grid Assembly. *AIAA Paper 2005-5117, 17th AIAA Computational Fluid Dynamics Conference*, Toronto, Ontario, Canada.
6. Liu, J. (2008). *Unsteady CFD Simulations of Moving Flow-Control Valves By An Unstructured Overset Grid Method*. Ph.D. Dissertation, Purdue University.
7. Karypis, G. and Kumar, C. (1998). *A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*. University of Minnesota, Department of Computer Science / Army HPC Research Center, Technical Report: 95-035.

Parallel Performance of ADPDIS3D – A High Order Multiblock Overlapping Grid Solver for Hypersonic Turbulence

B. Sjögreen[1], H.C.Yee[2], J.Djomehri[2], A.Lazanoff[2], W.D.Henshaw[1]

Keywords: Overset grids, Hypersonic flow, Finite difference scheme

1. Objective

The goal of this abstract is to evaluate the parallel performance of a newly developed compressible flow solver, ADPDIS3D, by reporting on recently performed benchmark computations with the solver on the new NASA supercomputer Pleiades. The essential feature of ADPDIS3D is that it combines the capabilities of a hypersonic flow solver for complex geometries with the capabilities of a solver for direct numerical simulation of turbulence.

2. Parallel flow solver and grid generator

ADPDIS3D is a time-accurate solver that is capable of solving a number of different equation sets for simulation of compressible fluid flow. The currently implemented models include the Euler/Navier-Stokes equations of standard compressible gas dynamics, the equations of non-ideal magnetohydrodynamics, and the equations of multi-species combustion with or without thermodynamic non-equilibrium models. The code is designed for simulation of hypersonic turbulent flows, including combustion, plasma, thermal and chemical non-equilibrium flows. ADPDIS3D approximates the flow equations by node centered finite difference discretizations on curvilinear grids. For simulation in complex geometries, ADPDIS3D uses composite overset grids generated by the Ogen [1] grid generator.

ADPDIS3D is based on low dissipative high-order accurate spatial methods that include limiting and filtering with flow sensors [5, 2, 6]. The idea of the method is to advance in time with a highly accurate base scheme, and to apply a nonlinear filter after each full time step to suppress Gibb's oscillations at discontinuities and other unphysical phenomena caused by locally non-smooth solution features. The filters consist of the dissipative portion of a high order shock-capturing method multiplied by a flow sensor, where the flow sensor switches on the dissipation only where needed. Any shock-capturing scheme can be used for shock-capturing dissipation. The preferred flow sensors in ADPDIS3D are based on a wavelet decomposition of the flow field.

The current version of ADPDIS3D contains central spatial base schemes of orders up to 14, adaptive nonlinear filters obtained from dissipation of WENO schemes of orders up to nine, and linear filters of orders up to 16. For the case of standard gas dynamics without shocks, entropy splitting of the inviscid flux derivative [4] can be employed to minimize the amount of numerical dissipation. For cross comparisons, a number of standard shock-capturing schemes are also implemented. ADPDIS3D advances the solution in time by Runge-Kutta time stepping, where the order of accuracy can be one, two, three, or four.

One extra complication for parallel computation with high order schemes is that the computational stencils are very wide, which leads to a large number of additional ghost points at the boundaries between processor blocks. Furthermore, ADPDIS3D uses summation-by-parts boundary modification

[1]Lawrence Livermore National Laboratory, Livermore, CA, 94551, USA. [sjogreen2,henshaw]@llnl.gov. Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-ABS-411008.

[2]NASA Ames Research Center, Moffett Field, CA 94035, USA. [Helen.M.Yee,Mohammad.J.Djomehri, Arthur.S.Lazanoff]@nasa.gov

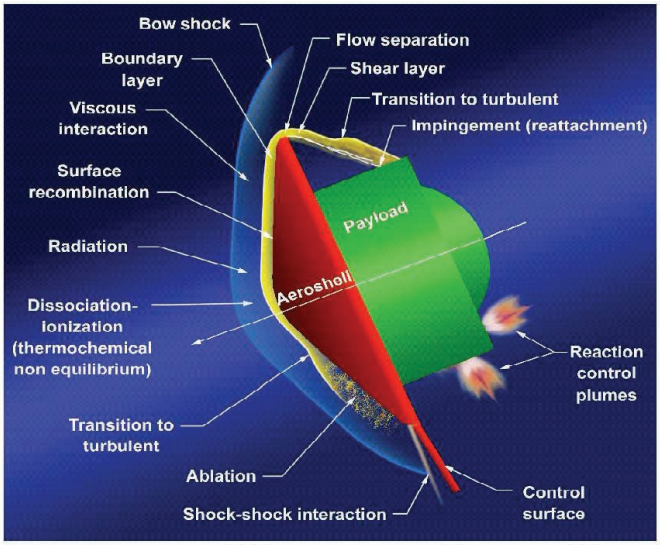


Figure 1: Flow phenomena encountered around an entry vehicle.

of the operators near the physical boundaries. When the number of processors increases, these boundary operators can make up a significant part of the computational domain in some of the processors. The load balancing algorithm used by ADPDIS3D takes these effects into account.

For parallel execution on overset grids, each component grid is evenly distributed on the total number of processors available. This gives perfect load balancing, but the amount of communication is larger than optimal. For the explicit time stepping used in ADPDIS3D, the communication cost is still only a small fraction of the total computation time. The approach is most efficient when the composite grid is made up of a few large component grids, because of low computation to communication ratio when component grids with very few grid points are distributed on a large number of processors. See Section 4 below for examples of performance.

The overlapping grid generator Ogen (part of the Overture framework developed at LLNL) is used to construct overlapping grids for ADPDIS3D. Given a set of structured curvilinear component meshes that cover a computational domain and overlap where they meet each other, the overlapping grid generation algorithm will determine how the grids should interpolate from one another. This process involves the cutting of holes where portions of grids are removed where they lie outside the computational domain. There are a number of important issues that must be addressed to make the grid generation process general, robust and automatic. The algorithm used by Ogen has been recently enhanced to run on distributed memory parallel computers. This speeds up the grid generation process and also permits very large grids to be generated.

3. Example computations

Often, hypersonic turbulent flows around re-entry space vehicles and space physics involve mixed steady strong shocks and turbulence with unsteady shocklets. Figure 1 illustrates a schematic of many of the possible steady and unsteady flow types. While sixth-order or higher order shock-capturing methods are appropriate for unsteady turbulence with shocklets, lower order shock-capturing methods

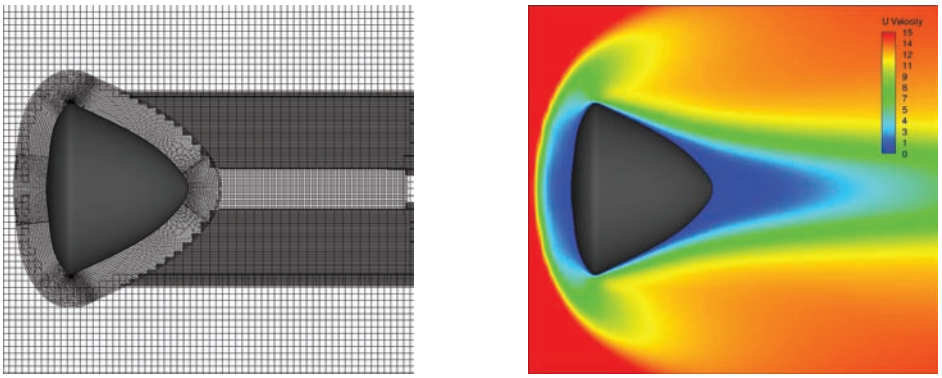


Figure 2: Grid and Mach number color levels in logarithmic scale for an Apollo-like CEV, Mach 16.

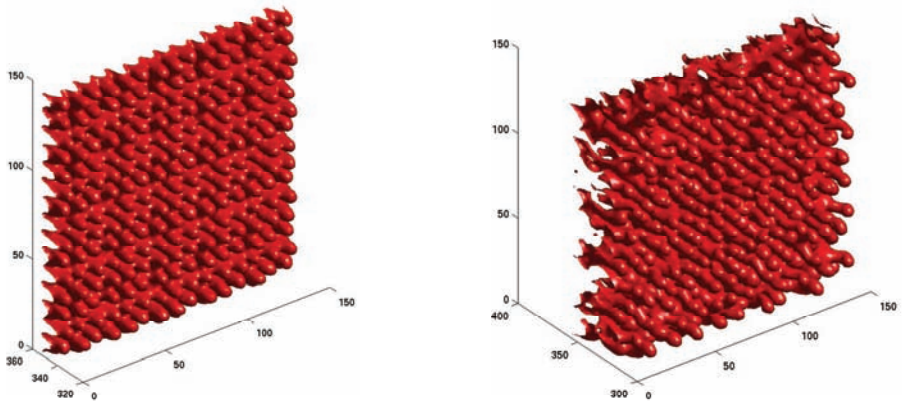


Figure 3: Richtmyer-Meshkov instability in Air/SF_6. Material interface before (left) and after (right) reshock. Iso-surface at air mass fraction 1/2.

are more effective for strong steady or nearly steady shocks in terms of convergence. In order to minimize the shortcomings of low order and high order shock-capturing schemes, ADPDIS3D allows different order of accuracy on different component grids. The two- and three-dimensional test cases reported in [3] illustrate that the overall error in high speed flow computations is reduced, even if high order schemes are used on only some of the component grids. The two examples shown below, in Figs. 2–3, demonstrate the unique capability of ADPDIS3D of both solving for flows in complex geometries, and to perform direct numerical simulations of turbulence.

The Apollo-like crew exploration vehicle (CEV) in Fig. 2 is computed on a six component overset grid system. Figure 2a shows a two-dimensional slice through the grids, and Fig. 2b shows computed Mach number color levels on the same slice. The free stream Mach number is 16 in this computation.

Figure 3, shows turbulent mixing by a Richtmyer-Meshkov instability. The initial condition is a tube with air to the left and SF_6 to the right of an interface. The tube is closed at the right end. A shock wave is sent in from the left. After passing through the air/SF_6 interface the shock wave is reflected at the end of the tube. The reflected shock passes through the interface and mixes the interface once more. Figure 3 shows the interface, represented as the iso-surface of $y_{air} = 1/2$, where y_{air} is the mass fraction of air in the mixture. The interface is given a small initial perturbation to trigger multi-

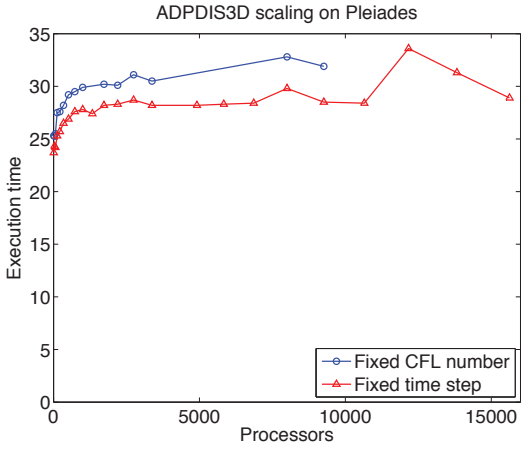


Figure 4: Execution time with weak scaling for a single grid computation. 20 time steps. Fixed time step (red) requires no global communication. Fixed CFL number (blue) computes the local time step and takes global minimum to compute.

dimensional effects. The increased mixing of the reflected shock can be assessed from Fig. 3, where the left subfigure shows the interface after the first shock interaction, and the right subfigure shows the interface after the interaction with the reflected shock.

4. Parallel performance

Figure 4 shows the weak scaling on Pleiades (NASA Ames supercomputer with 48,000 CPUs) for the Taylor-Green vortex problem in three space dimensions on a single block grid. The total execution time for a small number of time steps is measured when the number of grid points per processor is fixed at 216,000. The number of processors varies up to 15,625. The number of processors for the data points in Fig. 4 is always a cube, p^3, for integers $p = 2, \ldots, 25$, which means that not only is the number of points per processor constant, but also that the shape of data in each processor is always the same (a cube). The work load at the boundaries of the domain are identical in each processor, because the boundary conditions are periodic, Therefore, the number of arithmetic operations are exactly the same in each processor. Any deviation from a perfectly constant execution time must be attributed to differences in the network performance. It is not surprising that the performance is superior when the number of processors is small, because, in that case, the processors are closer to each other, in the same node, or in the same rack. Figure. 4 shows both the performance with fixed time step (red) and with fixed CFL number (blue). When the time step is fixed, no global communication is necessary, but when the CFL number is fixed, the size of the time step is computed before each time step. This entails more arithmetic operations and a global communication step to transmit the maximum stable time step to all processors. However, as shown by Fig. 4, the cost of the global communication, which theoretically should increase logarithmically with the number of processors, is not significant, at least up to 10,000 processors.

Figure 5 shows the strong scaling on Pleiades for the Apollo-like CEV geometry in Fig. 2, with 65 million grid points on six component grids. Figure 5 shows results with up to 1024 processors, which corresponds to approximately 58,000 points per processor. The performance started to seriously degrade, for larger number of processors. Figure 5a shows the execution time for 100 time steps vs. number of processors, and Fig. 5b displays the corresponding efficiency. The efficiency is

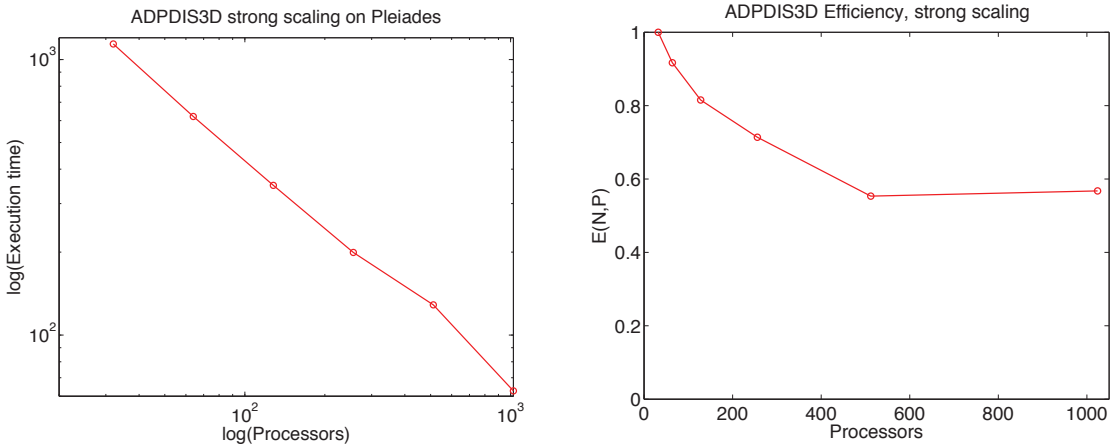


Figure 5: Strong scaling performance for an overset grid system of 60 million grid points. a) (left) Execution time in seconds for 100 steps, logarithmic scale. b) (right) Efficiency (right) for the same data.

normalized to be one at the first data point, which was obtained with 32 processors. Here the nontrivial geometry, grids with cut out holes, and the global communication step required for the interpolation between overset grids are difficulties that are not present in the previous weak scaling example.

5. Acknowledgments

The financial support from the NASA Fundamental Aeronautics (Hypersonic) program for the second author is gratefully acknowledged. The authors thank Nagi Mansour for providing Figure 1.

References

[1] W.D.Henshaw, *Ogen: An Overlapping Grid Generator for Overture*, Report UCRL-MA-132237, Lawrence Livermore National Laboratory, (1998).

[2] B. Sjögreen and H. C. Yee, *Multiresolution Wavelet Based Adaptive Numerical Dissipation Control for Shock-Turbulence Computation*, RIACS Technical Report TR01.01, NASA Ames research center (Oct 2000); also, J. Scient. Computing, **20** (2004) 211–255.

[3] B. Sjögreen and H. C. Yee, *Variable High Order Multiblock Overlapping Grid Methods for Mixed Steady and Unsteady Multiscale Viscous Flows*, Commun. Comput. Phys., **5** (2009) 730-744.

[4] H.C. Yee, M. Vinokur, M., and M.J. Djomehri, *Entropy Splitting and Numerical Dissipation*, J. Comput. Phys., **162** (2000) 33–81.

[5] H.C. Yee and B. Sjögreen, *Development of Low Dissipative High Order Filter Schemes for Multiscale Navier-Stokes/MHD Systems*, J. Comput. Phys., **225** (2007) 910–934.

[6] H.C. Yee and B. Sjögreen, *Adaptive Filtering and Limiting in Compact High Order Methods for Multiscale Gas Dynamics and MHD Systems*, Computers & Fluids., **37** (2008) 593–619.

Efficiency enhancement of an unstructured CFD-Code on distributed computing systems

Thomas Alrutz*, Christian Simmendinger, Thomas Gerhold*****

* T-Systems Solution for Research GmbH, 37073 Göttingen, Germany
(e-mail: Thomas.Alrutz@t-systems-sfr.com)

** T-Systems Solution for Research GmbH, 70569 Stuttgart, Germany
(e-mail: Christian.Simmendinger@t-systems-sfr.com)

*** DLR, Institute of Aerodynamics and Flow Technology, 37073 Göttingen, Germany
(e-mail: Thomas.Gerhold@dlr.de)

Abstract: In this paper we describe several improvements with respect to parallel efficiency and scalar performance of unstructured CFD-Codes. The optimizations are generic in the sense that they are applicable to any CFD-Code which works on unstructured grids. The code modifications were sucessively introduced in the DLR TAU-Code and the results of the optimizations are demonstrated on two different aircraft configurations. We give an outlook of further improvements with respect to upcoming multicore computing architectures.

Keywords: unstructured CFD, distributed computing, performance optimization

1 INTRODUCTION

Computational Fluid Dynamics (CFD) is a key technology for the aircraft industry. Growing use of CFD in the aircraft design and the aerodynamic loads process demands increased simulation capabilities and simulation capacity. Thus, optimization of run time performance and parallel scalability is one important aspect to be faced among others like algorithmic acceleration. This is especially true for the current multi-core computer systems, because more and more parallel processes will be available for single simulations. The challenge will be to keep and even increase the parallel efficiency for massive parallel simulations on future many-core-computers.
This paper describes several improvements with respect to parallel efficiency and scalar performance of an unstructured CFD-Code. The optimizations are generic such that any unstructured CFD-Code could in principle benefit from the proposed optimizations. We have introduced the modifications in the DLR TAU-Code and used two aerodynamic configurations to demonstrate the results.

1.1 The TAU-Code

The DLR TAU-Code is a Finite-volume Euler/Navier-Stokes solver working on unstructured hybrid grids. The code is composed of independent modules: Grid partitioner, preprocessing, flow solver, grid adaptation and other modules. The different modules of TAU, described in more detail in [4], can both be used as stand-alone tools with corresponding file I/O or within a Python scripting framework which allows also for inter-module communication without file-I/O, i.e. using common memory allocation. The Preprocessor generated a dual grid stored in an edge based data structure and coarse dual grid levels using an agglomeration technique in order to allow for multigrid acceleration. The flow solver is a three-dimensional finite volume scheme for solving the unsteady Reynolds-Averaged Navier-Stokes equations.

2 PERFORMANCE ENHANCEMENT

We have optimized the entire simulation workflow. We will describe the changes for each element of this workflow in the following sections.

2.1 Grid Partitioning

The parallelization of the flow solver is based on a domain decomposition of the computational grid. The standard grid partitioner used a recursive bisection algorithm to compute the desired number of grid domains. However, with a growing number of domains, the recursive bisection algorithm shows its limitations with respect to parallel efficiency (see figure 2 (b)). The main bottleneck of this algorithm is the way how the grid domains are load balanced. Due to the recursive nature of the bisection algorithm, the partitions acquire a growing inbalance for each sucessive split. Additionally the number of cut-edges and domain neighbours has not been optimal. In order to overcome these limitations for large numbers of CPU-cores, we have used the *CHACO* toolkit [2] to calculate the partitions. In doing so, we also have changed the calculation of the computational load. In the old formulation only the number of connecting edges was used to model the computational cost of a grid partition. The new formulation additionally takes into account the weighted number of grid points.

2.2 Preprocessing

In order to achieve high scalar efficiency on cache-based architecures, we needed to block data in the caches. To that end we have maximized both spatial and temporal data locality.

2.2.1 Decomposition

In a first step we have decomposed all MPI-Domains into several sub-domains (colors). The sub-domais are equally sized with respect to the number of data points they contain.
The size of these sub-domains then is chosen such that all the data is blocked in the L2-cache. In order to achieve a high temporal data locality, we have re-organized the solver algorithm in the following fashion: Instead of computing e.g. all flux contributions sequentially over the entire MPI-domain,

```
1 add_inner_inviscid_central(grid, ...);
2 add_inner_viscous_fluxes(grid, ...);
3 turb_diffusion_fluxes(grid, ...);
4 turb_sources_implicit(grid, ...);
```

we instead loop over all sub-domains. Within each sub-domain we compute all flux contributions,

```
1 for(color = grid->gedat->fcolor; color != NULL; color = color->succ){
2   add_inner_inviscid_central(grid, faces_of_color, ...);
3   add_inner_viscous_fluxes(grid, faces_of_color, ...);
4   turb_diffusion_fluxes(grid, faces_of_color, ...);
5   turb_sources_explicit(grid, faces_of_color, ...); }
```

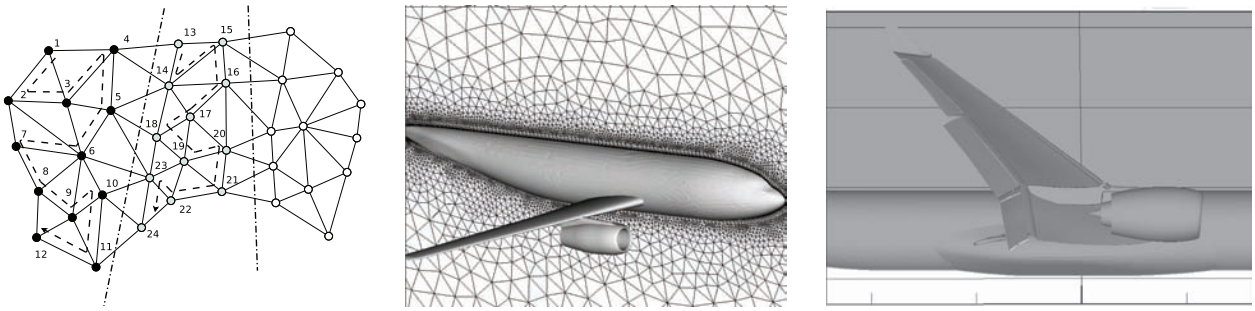
where all loops run over the faces of the color. We use the following convention: A face belongs to a color if the attached point with the lower point number belongs to the color.

2.2.2 Point Order

While a high temporal data locality already achieves a substantial speedup, we additionally have maximized the spatial data locality. To that end we have renumbered the points along a space filling Hilbert Curve. For an example we refer to figure 1 (a). For every domain the Hilbert Curve starts at the point which is closest to the top left corner. We then continously seek a neighbour of the previous point such that this next point has a maximum of faces connecting it to the points we already have renumbered. In case of several neighbour points with an equal number of faces connected to previous points, we again chose the point which is closest to the top left corner.

2.2.3 Face Order

We also have modified the sequence of faces, in which the solver computes e.g. the flux contributions. We have found that a near optimal sequence can be achieved by firstly re-ordering the points and then sorting the faces with respect to the attached point numbers.



(a) sub-domain decomposition (b) DLR F6 2×10^6 grid points (c) Highlift 13.6×10^6 grid points

Figure 1: Grid decomposition (a) and aerodynamic configurations F6 (b) and highlift (c).

2.3 Solver

In the flow solver we extended the concept of sub-domains by additionally fusing point and face loops. Instead of subsequently initializing the gradients over the entire sub-domain, computing all gradients and then normalizing by dividing with the control volume

```
for(pnt = 0; i < npoints; pnt++)
  initialize_gradients(pnt, ...);
for(face = 0; face < nfaces; face++)
  compute_gradients(face, ...);
for(pnt = 0; i < npoints; pnt++)
  normalize_gradients(pnt, ...);
```

we additionally have fused point and face loops.

```
for(color = grid->gedat->fcolor; color != NULL; color = color->succ) {
    initialize_gradients(points_of_color, ...);
    compute_gradients(faces_of_color, ...);
    normalize_gradients(points_of_color, ...); }
```

With this method we were able to extend the concept of sub-domains over large parts of the solver. We have used the resulting temporal and spatial data locality to introduce pre-computed values for a number of variables. E.g. we take the calculation of a distance between two points

```
dx   = xx[p1][0] - xx[p0][0];
dy   = xx[p1][1] - xx[p0][1];
dz   = xx[p1][2] - xx[p0][2];
rld2 = 1.0 / (dx * dx + dy * dy + dz * dz)
```

and replace the computation of $1/(dx^2 + dy^2 + dz^2)$ by

```
rld2 = rdist[face] * rdist[face];
```

with $rdist = 1/\sqrt{dx^2 + dy^2 + dz^2}$ is determined in the initialization phase of the solver.
This replacement is only possible due to the performance gain from the high cache efficiency and the reduced number of compute cycles, which compensates the additional load.

3 RESULTS

To demonstrate the achieved performance gain due to the optimization of the code we performed a RANS calculation with two aerodynamic configurations. One is the well known DLR F6 configuration (figure 1 (b)) with 2×10^6 grid points and 5×10^6 volume elements already presented in [3] and the other is a highlift configuration (figure 1 (c)) with 13.6×10^6 grid points and 35×10^6 volume elements. For both configurations we used the explicit Runge-Kutta flow solver from the TAU-Code with the following settings (see also [1]):

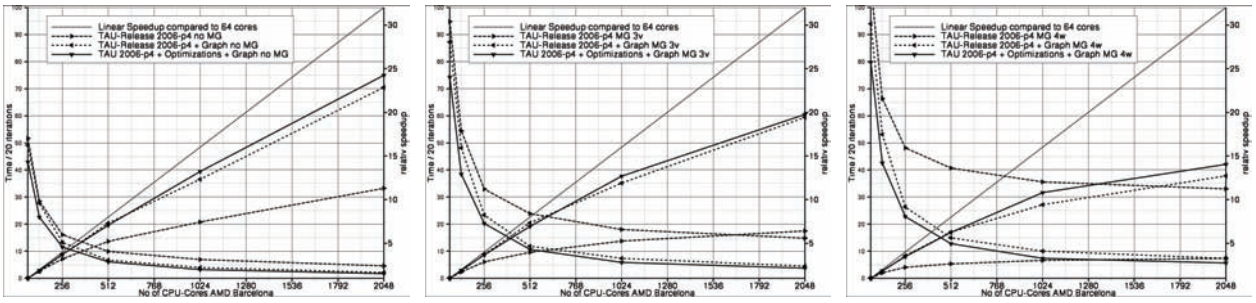
```
1 Turbulence model: Spalart Allmaras with Edwards modification
2 Solver type: central scheme
3 Number of Runge–Kutta steps: 3
4 Multigrid: 4w cycle (highlift also 3v and no multigrid)
5 Number of iterations: 50 (DLR F6), 200 (highlift)
```

The DLR F6 configuration was used to show the evolution of the TAU-code performance due to the source code optimizations over the last decade (see table 1).

Table 1: The improvement of the TAU-Code performance over the last years in MFlops[1].

	2.4 GHz Opteron Single Core				2.6 GHz Opteron Dual Core			
CPUs	2002	% Peak	2004	% Peak	2006	% Peak	Optimized	% Peak
4	2547	13.26	2969	15.46	3730	17.93	4278	20.56
64	33245	10.82	40781	13.27	41613	12.50	56327	16.92
128	48549	7.90	63720	10.31	62166	9.33	94987	14.26

Whereas the highlift configuration was used to demonstrate the efficiency gain and scaleability of the TAU-Code when using large numbers of CPU-cores (see figure 2).



(a) No multigrid (b) 3v multigrid (c) 4w multigrid

Figure 2: Parallel performance of TAU-Code versions on [5] with the highlift configuration.

4 CONCLUSION AND OUTLOOK

We have increased scalar and parallel effciency of the unstructured CFD code TAU by a substantial margin. All optimzations are generic in the sense that they are applicable to any CFD-Code which works on unstructured grids. We are currently investigating the possibility to achieve even higher speedups by eliminating the indirect access by means of a L2 cache local workarray. The underlying reason is that the indirect access, which is typical for all unstructured CFD solvers, prevents a SIMD vectorization of the code.

REFERENCES

[1] Thomas Alrutz. Investigation of the parallel performance of the unstructured DLR-TAU-Code on distributed computing systems. In A. Deane et al., editor, *Parallel Computational Fluid Dynamics*, pages 509–516, Proceedings of the 16th Parallel CFD Conference, College Park, MD, USA, May 24-27 2005. Elsevier.

[2] Bruce Hendrickson and Robert Leland. Chaco: Software for Partitioning Graphs. *http : //www.sandia.gov/bahendr/chaco.html*.

[3] Norbert Kroll, Thomas Gerhold, Stefan Melber, Ralf Heinrich, Thorsten Schwarz, and Britta Schöning. Parallel Large Scale Computations for Aerodynamic Aircraft Design with the German CFD System

[1] Calculated with the DLR F6 configuration on Infiniband Linux-clusters.

MEGAFLOW. In *Proceedings of Parallel CFD 2001*, 2001. Egmond aan Zee, The Netherlands, May 21-23.

[4] Dieter Schwamborn, Thomas Gerhold, and Ralf Heinrich. The DLR TAU-Code: Recent Applications int Research and Industry. In P. Wesseling, E. Onate, and J. Periaux, editors, *Proceedings, ECCOMAS CFD 2006 CONFERENCE*, September 5-8 2006.

[5] Texas Adavanced Computing Center (TACC). Ranger, 2008. *http* : *//www.tacc.utexas.edu/resources/hpcsystems/*.

Parallel Poisson solver for revolved unstructured grids. DNS of the flow around a sphere at $Re = 3700$

R. Borrell[a], O. Lehmkuhl[b,a], I. Rodríguez[a], C.D. Pérez Segarra[a], A. Oliva[a]

[a] *Centre Tecnològic de Transferència de Calor, Technical University of Catalonia (UPC)*
ETSEIAT, c/ Colom 11, 08222 Terrassa, Spain
[b] *TERMO FLUIDS S.L.,c/ Magi Colet 8, 08204 Sabadell, Spain*

Abstract

In this paper a parallel direct Poisson solver for DNS simulation of turbulent flows in domains of revolution using unstructured meshes is presented. It is based on a Fourier diagonalization in the azimuthal direction and a Schur complement based decomposition on the other two directions. Numerical experiments carried out on IBM MareNostrum supercomputer to test the robustness and efficiency of the algorithm are presented. This solver is being used for a DNS of a turbulent flow around a sphere. The results of the initial case of study at $Re = 3700$ are presented.

Key words:
parallel direct Poisson solver, Direct Numerical Simulation (DNS), Schur complement, FFT, unstructured meshes, domains of revolution, flow around sphere

1. Introduction

Direct Numerical Simulation (DNS) of turbulent flows are rarely used for 'real applications'. The main reason is that the mesh size necessary to avoid subgrid scales is proportional to $Re^{9/4}$, and the time step scales with $Re^{1/2}$. Altogether, this means that the computational complexity scales with $Re^{11/4}$. However, DNS are of high interest for the study of the physics of turbulent flows because the numerical results are obtained without modelling any term of the Navier-Stokes equations. Besides, DNS has become very important for the improvement and validation of new turbulence models.

In the numerical algorithm used for DNS, the solution of a Poisson equation, which arises from the incompressibility constraint and has to be solved at least once at each time step, is usually the main bottleneck in terms of RAM memory and CPU time requirements. In this context, efficient and scalable algorithms for the solution of the Poisson equation are of high interest.

A parallel Schur-Fourier Decomposition algorithm, for the solution of the Poisson equation discretized in domains of revolution with extruded unstructured meshes, is proposed in the present work. This method has been used before to carry out DNS for a differentially heated cavity at high Rayleigh numbers using *Cartesian* grids [1]. The goal of the present work is to extend it for revolved unstructured grids.

2. Mathematical and numerical method

The Navier-Stokes and continuity equations can be written as

$$\Omega \frac{\partial \boldsymbol{u}}{\partial t} + \mathsf{C}(\boldsymbol{u})\,\boldsymbol{u} + \mathsf{D}\boldsymbol{u} + \Omega \mathsf{G}\boldsymbol{p} = \boldsymbol{0} \quad (1)$$

$$\mathsf{M}\boldsymbol{u} = \boldsymbol{0} \quad (2)$$

where $\boldsymbol{u} \in \mathbb{R}^m$ and $\boldsymbol{p} \in \mathbb{R}^q$ are the velocity vector and pressure, respectively. The matrices $\mathsf{C}(\boldsymbol{u})$, $\mathsf{D} \in \mathbb{R}^{m \times m}$ are the convective and diffusive operators, respectively. Note the $\boldsymbol{u}$-dependence of the convective operator (non-linear operator). Finally, $\mathsf{G} \in \mathbb{R}^{m \times q}$ represents the gradient operator, and the matrix $\mathsf{M} \in \mathbb{R}^{q \times m}$ is the divergence operator.

For the discretization of the momentum equation (1), a second order backward difference scheme for the time derivative term, a fully explicit second-order one-leg scheme [2] for the convective and diffusive terms, and a first-order backward Euler scheme for the pressure gradient are used.

Our spatial discretization schemes are conservative, i.e., they preserve the kinetic energy equation, which allow good stability properties even at high Reynolds numbers and with coarse meshes. These conservation properties are held if and only if the discrete convective operator is skew-symmetric ($\mathsf{C}_c(\boldsymbol{u}_s) = -\mathsf{C}_c^*(\boldsymbol{u}_c)$), if the negative conjugate transpose of the discrete gradient operator is exactly equal to the divergence operator ($-(\Omega_c \mathsf{G}_c)^* = \mathsf{M}_c$), and if the diffusive operator D_c is symmetric and positive-definite.

To solve the velocity-pressure coupling, a classical fractional step projection method [2] is used,

$$\boldsymbol{u}_c^p = \boldsymbol{u}_c^{n+1} + \mathsf{G}_c \tilde{\boldsymbol{p}}_c \tag{3}$$

where the pseudo-pressure is $\tilde{\boldsymbol{p}}_c = \boldsymbol{p}_c^{n+1} \Delta t / (\beta + 1/2)$, and $\boldsymbol{u}_c^p$ is the predicted velocity. The discrete Poisson equation for $\tilde{\boldsymbol{p}}_c$ is obtained by taking the divergence of (3) and after applying the incompressibility condition,

$$\mathsf{L}_c \tilde{\boldsymbol{p}}_c = \mathsf{M}_c \boldsymbol{u}_c^p \tag{4}$$

where discrete laplacian operator $\mathsf{L}_c \in \mathbb{R}^{q \times q}$ is, by construction, a symmetric positive-definite matrix ($\mathsf{L}_c \equiv \mathsf{M}_c \Omega^{-1} \mathsf{M}_c^*$). Once the solution is obtained, $\boldsymbol{u}_c^{n+1}$ results from the correction

$$\boldsymbol{u}_c^{n+1} = \boldsymbol{u}_c^p - \mathsf{G}_c \tilde{\boldsymbol{p}}_c \tag{5}$$

The origin of checkerboard problem is related with the unrealistic components of cell-centred velocity field that projection matrix cannot eliminate. The most common way to tackle this problem is to slightly change some of the operators used in this projection step. Such modifications are generally introducted in the divergence operator M as in the traditionally used Momentum Weighted Interpolation Method [3]. Our approach is to change the equation (5) with a Least Squares gradient reconstruction method. Doing so, the properties of the pressure gradient operator are slightly changed and the system is conditioned without any modification in the convective operator.

2.1. Solving the Poisson equation

As a result of the discretization, the Poisson equation obtained is:

$$\mathsf{L}_c x = b \tag{6}$$

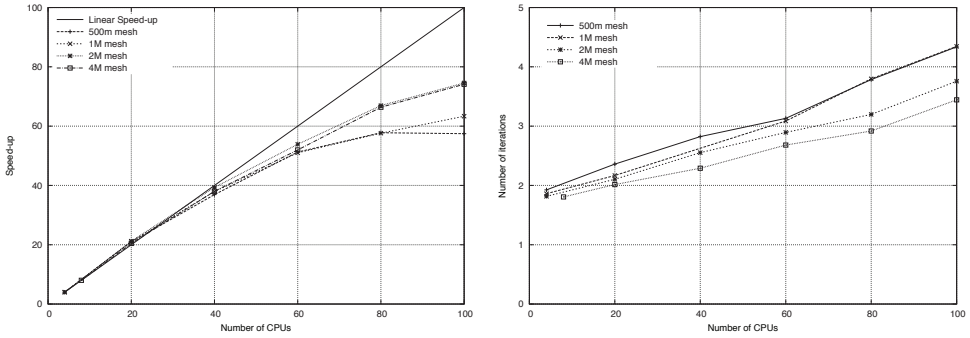
where $\mathsf{L}_c \in \mathbb{R}^{q \times q}$ is the Laplacian operator derived in previous section, and $x, b \in \mathbb{R}^q$ are the solution and right-hand side term respectively. Taking into account the geometry of the domain, the mesh is generated by revolving a 2D unstructured mesh with a constant step $2\pi/N_{per}$. With this grid, the linear dependences in the azimuthal direction are given by circulant matrices. This allows to solve the Poisson equation by means of a Fourier diagonalization method. In the spectral space (diagonalization space), the Laplacian operator has the form

$$\mathsf{L}_{freq} = \bigoplus_{k=1}^{N_{per}} \mathsf{L}_{f_k}. \tag{7}$$

Thus, it is decoupled in N_{per} $two-dimensional$ systems:

$$\mathsf{L}_{f_k} x_{f_k} = b_{f_k} \qquad k = 1, ..., Nper; \tag{8}$$

reducing dramatically its arithmetical complexity and the RAM memory requirements to solve it. Notice that the DFT necessary to transform the right-hand side term from physical to the spectral space ($b \mapsto \bigoplus_{k=1}^{N_{per}} b_{f_k}$) and, after the system is solved, the solution backwards to physical space ($\bigoplus_{k=1}^{N_{per}} x_{f_k} \rightarrow x$), can be carried out by means of a FFT algorithm that has $O(nlogn)$.



(a) (b)

Figure 1: Numerical experiments on MareNostrum supercomputer. a) *Strong* speedup of DSD method to solve *two – dimensional* systems (8). b) Number of iterations of local Sparse Cholesky preconditioned CG with the time spent by DSD.

The N_{per} *two – dimensional* systems (8) are solved by means of a Direct Schur complement based Decomposition (DSD) method [4, 5]. This method is based on the explicit calculation of the Schur complement matrix of the partitioned system L_{f_k}, and its direct solution. Although the large RAM memory requirements and pre-processing time, on the solution step this methodology can be very fast compared with iterative solvers. In this particular case, since the systems to be solved are *two – dimensional*, the size of the Schur complement matrix grows linearly with the mesh size. Meshes with the initial extruded plane of size up to 4 million nodes are considered in this paper. On the other hand, the pre-processing is carried out only once because the matrix L is constant during the time integration.

Different numerical experiments have been carried out on IBM MareNostrum supercomputer to test the efficiency of the algorithm. In Fig. 1(a) the *strong* speedup of DSD solver with meshes of sizes: 0.5, 1, 2 and 4 millions nodes is plotted. The main cause of the speedup limitation of this method is the growth of the Schur Complement matrix with the number of CPU. Note that when the size is increased the speedup improves because the percentage of time spent in the solution of systems grows.

In Fig. 1(b), it is showed the iterations that can be performed with local Sparse Cholesky preconditioned CG, with the time needed to solve the system using DSD. Note that the number of iterations grows with the number of CPU, this is because the speedup of CG-Cholesky method is better. However as the size of the diagonal blocks decreases with the number of CPU the method needs more iterations to reach convergence. These results mean that given a mesh, a group of CPU, and a required accuracy; if the number of iterations needed by the iterative solver are higher than the value plotted, its more efficient to use the direct solver. In general, lower frequencies of system (8) are easier to solve for the iterative solver. This is because its systems are more diagonal dominant and the initial field, given by the solution in the previous time-step, is more accurate. In [6] different strategies are used for the different 2D systems depending on how well-conditioned they are.

Apart from the parallelization of $L_{f_k} x_{f_k} = b_{f_k}$ systems solved by P_{freq} processors; the set of frequencies is partitioned into P_{spec} subsets. Thus, the total number of processors used in the solution is $P_{total} = P_{spec}.P_{freq}$. In the absence of an efficient distributed memory FFT, data are replicated and a sequential FFT algorithm is used. This makes necessary to perform an alltoall communication before applying FFTs, and its the main limitation for the speedup produced by

the frequency partition.

In Fig. 2, is showed the *weak scalability* in the the azimuthal direction. The problem considered is the Poisson equation discretized in different revolved unstructured grids. The sizes of the revolved *two − dimensional* meshes are: 0.5, 1 and 2 million nodes respectively. For each point on a line, the size of the discretization in the periodic direction (N_{per}) is twice than the previous point. The value of N_{per} for the starting point of the lines is 16, thus for the next points is 32, 64 and 128 respectively. The load for CPU is kept constant, the number of CPU's used in each point are 80, 160, 320 and 640 respectively. The times are normalized by the value on the starting point.

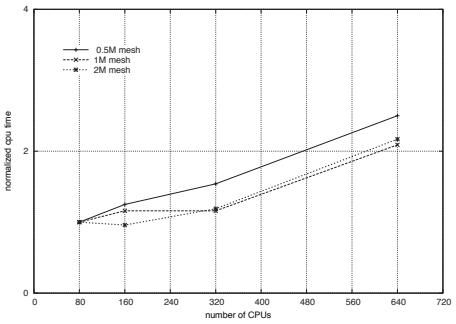
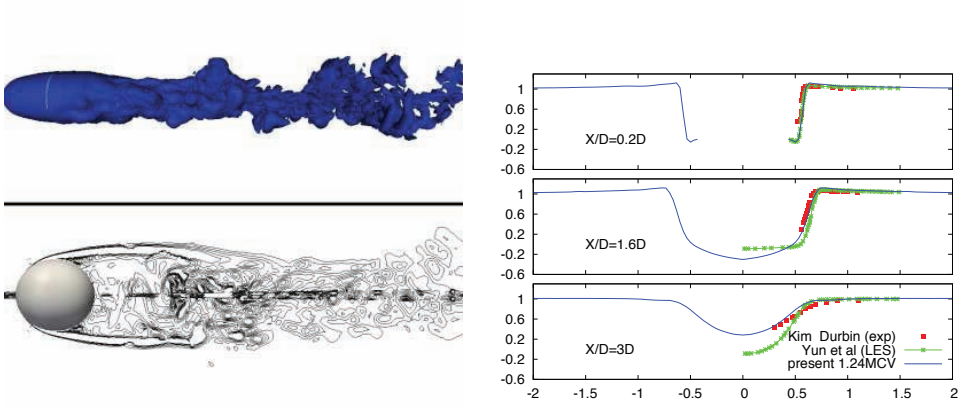


Figure 2: *Weak scalability* in the the azimuthal direction.

3. DNS around a sphere at $Re = 3700$

Numerical simulations of the flow around a sphere are performed at $Re = 3700$, where Reynolds number is defined in terms of the free-stream velocity U and the sphere diameter D. Solutions are obtained on a computational domain of dimensions [-5D,20D]; [0,5D]; [0,2π], where the sphere is located at $x = 0, y = 0$. The govering equations are solved on an unstructured mesh generated from the extrusion around the axis of a two-dimensional unstructured grid in a (x,y) plane.



(a) (b)

Figure 3: Illustrative results. a) Visualization of the vortical structures (top) and the contours of the instantaneous azimuthal vorticity (bottom), both obtained with the finnest grid (2.7 MCV). b) Averaged Streamwise velocity at three locations in the wake. Comparison of the results obtained (1.24 MCV mesh) with Kim and Durbin [7] and Yun et al. [8].

The boundary conditions at the inflow consist of a uniform velocity (u,v,w)=(1,0,0). Constant velocity (u,v,w)=(1,0,0) is also prescribed at the outlet boundaries except for the downstream one where a pressure based condition is used. No-slip conditions on the sphere are imposed.

In order to evaluate the influence of the grid resolution, calculations are carried out using two different grids . The coarse mesh is composed of 1.24 M (19351 x 64 planes) of control volumes, while the finnest one has 2.7 M CV (42142 x 64 planes).

Some illustrative results obtained are depicted in Figure 3. Vorticity structures in the near wake obtained with the finnest grid are plotted in Fig. 3(a). Furthermore, streamwise velocity at different locations in the wake, in comparison with literature available results, is plotted in figure 3(b). As can be seen, results obtained with both meshes are very promising as they are able to predict the mean flow quantities successfully.

Acknowledgements

This work has been financially supported by *Termo Fluids S.L.* and the *Ministerio de Educación y Ciencia*, Spain, (Project: "Development of high performance parallel codes for the optimal design of thermal equipments". Contract/grant number ENE2007-67185).

Calculations have been performed on the JFF cluster at the CTTC and on the IBM MareNostrum supercomputer at the Barcelona Supercomputing Center. The authors thankfully acknowledge these institutions.

References

[1] F. X. Trias and M. Soria and A. Oliva and C. D. Pérez-Segarra. Direct numerical simulations of two- and three-dimensional turbulent natural convection flows in a differentially heated cavity of aspect ratio 4. *Journal of Fluid Mechanics*,586:259–293,2007.

[2] R. W. C. P. Verstappen and A. E. P. Veldman. Symmetry-Preserving Discretization of Turbulent Flow. *Journal of Computational Physics*, 187:343–368, May 2003.

[3] C. M. Rhie and W. L. Chow. Numerical Study of th Turbulent Flow Past an Airfol with Trailing Edge Separation. *AIAA Journal*, 21:1525–1532, 1983.

[4] Saad Y. et al. Distributed Schur complement techniques for general sparse linear systems. *SIAM Journal of Scientific Computing*, 21(4):1337–1356, 1999.

[5] R. Borrell, O. Lehmkuhl, M. Soria, and A. Colomer, Oliva. Schur complement methods for the solution of Poisson equation with unstructured meshes. In *Parallel Computational Fluid Dynamics*, 2007.

[6] A. Gorobets and F.X. Trias and M. Soria and A. Oliva. A scalable parallel Poisson solver for three-dimensional problems with one periodic direction. *Computers and Fluids*, under revision.

[7] Kim H.J. and P.A. Durbin. Observations of the frequencies in a sphere wake and of drag increase by acoustic excitation. *Physics of Fluids*, 31(11):3260–3265, 1988.

[8] Yun G., Kim D., and Choi H. Vortical structures behind a sphere at subcritical Reynolds numbers. *Physics of Fluids*, 18, 2006.

DESIGN OPTIMIZATION

Parallel performance of CFD applications and the ubiquitous need for HPC with high fidelity, multidisciplinary analysis and optimization (MDO)

Mark Kremenetsky* and Srinivas Kodiyalam**

** Silicon Graphics Inc. (SGI), Sunnyvale, CA 94085, USA*
(Email: mdk@sgi.com)
*** Silicon Graphics Inc. (SGI), Sunnyvale, CA 94085, USA*
(Email: skodiyal@sgi.com)

Abstract: We discuss possible ways to improve CFD application software performance by highlighting key features of the HPC clusters, including, multi-core processor performance, host channel adapters (HCA), multi-rail networks, and message passing (MPI) implementations and more as well as briefly highlight the ever-growing need for HPC for solving computing intensive MDO problems.

Keywords: HPC, CFD, MDO, clusters, multi-core, multi-rails, MPI, parallel scalability

1.0 INTRODUCTION

Parallel processing on high performance computing environments has enabled much larger and complex CFD problems to be addressed [1, 2]. It is noteworthy that while High Performance Computing (HPC) has indeed facilitated solutions to many of the complex CFD simulations and the growth in the usage of CFD, the application of most detailed approaches (such as, direct numerical simulations (DNS) of the Navier-Stokes equations) for industry standard vehicle configurations with typical operating conditions are still beyond today's computing capability.

It is well known that the cost of software development exceeds hardware upgrade costs in later years of the product life cycle. Besides, a "simplistic" usage of the newer generation hardware does not provide the expected or desired level of performance improvement and very often the engineer or scientist is unaware of the possible hardware options that can be employed to improve performance. This is simply due to the information gap between the engineering users, who are typically experts in their technical disciplines/applications, and the hardware specialists who are well versed in the hardware usage.

In this paper we will discuss possible ways to improve the application software performance (i.e. CFD codes) by highlighting key features of the HPC clusters, including, multi-core processor performance, host channel adapters (HCA), multi-rail networks, message passing (MPI) implementations and such as well as the ever-growing need for HPC for solving computing intensive MDO problems with high fidelity analysis codes, such as CFD and Structures.

2.0 PERFORMANCE OF CFD APPLICATIONS

Majority of the CFD applications, with few exceptions, is implemented in a distributed memory paradigm and rely heavily on the use of explicit message passing technology (MPI). The driving force behind such an approach is the desire to reach global portability over all possible computer architectures. The way the cluster nodes are connected together has a significant influence on the overall application performance, i.e. the cluster interconnect is critical to the efficiency and scalability of the entire cluster, as it needs to handle the I/O requirements from each of the CPU cores without imposing networking overhead on the computational performance. In multi-core processor architectures, the performance and scalability bottlenecks have shifted to I/O and memory configurations.

2.1 Performance on a single node

2.1.1 Multi-core processor performance: For a very long time, the primary source of performance improvement in new processors was expected from a higher clock rate but this feature did not fully address the performance expectations. Chip designers not only face physical layout problems in further increasing the clock rate but also other challenges including specific requirements of application codes such as those related to memory referencing. The latter includes memory latency, cache utilization and memory bandwidth. Application codes require well balanced processors – not only pure computational power but also mechanism by which the data is delivered. Classification of an application code/algorithm based on bandwidth requirements is well known with the two most popular categories being the bandwidth hungry and latency driven algorithms. The typical measure of a bandwidth

driven algorithm is the well known Triad benchmark while the second category will include any algorithm that relies on sparse data sets. However, this is not a very accurate classification and in reality all computational algorithms are bandwidth hungry. The difference is in the time pattern of memory requests- the first group shows a continuous stream of memory transfer while with the second category we have a discontinuous pattern where high bandwidth areas look like spikes on time axis. This means that practically all algorithms will benefit from processors with a higher memory bandwidth. For multi-core processors, it becomes more complicated because we have to pay attention to the complete data path from memory to cache to computational units.

The following example is using FLUENT CFD application code, from ANSYS Inc., based on unstructured grid discretization that in turn defines the sparse nature of the main data structures. This example is used to illustrate the multi-core processor performance with 2 Intel Clovertown sockets in a node. The Intel Clovertown processor is a quad core processor that consists of two dies, each of those dies contains 2 computational cores and one shared L2 cache. We ran 4 threads computation on a single node that contains 2 Clovertown sockets using various threads to core allocation schemes and the results are provided below:

Table 1: Threads to core allocation

	4 threads/socket; shared L2 cache; 1 FSB	2 threads/socket; shared L2 cache, 2 FSB	2 threads/socket; non shared L2 cache; 2 FSB
Bandwidth (MB/sec)	550/1.0	504/1.09	426/1.29
Time (sec)	46/1.0	39/1.18	31/1.48

The FSB for this particular chipset ran at 1333 MHz so hardware bandwidth is about 10.6 GB/s. As seen, the average bandwidth consumed by the CFD application is much lower then available hardware bandwidth but nevertheless measured timings are quite different. Our interpretation is that using the optimal thread allocation with non-shared L2 cache results in a much lower number of memory requests and as such the memory consumption spike is much lower and that leads to a better application performance over all.

The bandwidth related problem that are specific to the current generation of multi-core processors are extensively resolved with the new Intel quad-core processor, Nehalem. Nehalem is one of the new processors that provide for a better balance between computational units and the memory path characteristics.

1. The Nehalem socket has a memory controller device directly on the die that allows doubling the available bandwidth. Depending on the memory type, the hardware bandwidth is in the range of 20GB/s - 35 GB/s;
2. Each core has its own L1 and L2 cache to substantially decrease the number of stalls in a data path;
3. The data pre-fetch algorithm for L2 and L3 caches has been reworked to achieve more effective data load.

The above improvements have contributed to substantial improvement in application performance over the previous generation of processors.

Following is a comparison of FLUENT standard benchmark suite performance on a node with 2 Harpertown quad-core sockets/processors running at 3GHz and a second node with 2 Nehalem quad-core processors at 2.66GHz. The performance metrics used in this study is the same rating number used by ANSYS that represents the number of jobs completed in a 24-hour time frame. Hence, a higher number represents better performance.

Table 2: Comparison of Intel Harpertown to Nehalem processors

# of cores (threads)	Aircraft (2M cells) Harpertown/Nehalem	Eddy (417K) H/N	Sedan (4M) H/N	Turbo (500K) H/N
1	94/134	110/138	61/105	444/617
2	183/266	218/268	122/213	866/1201
4	275/493	389/499	185/397	1400/2223
8	312/812	540/881	217/676	1835/3927

As seen from the above results, Nehalem processor with a lower clock rate performs significantly better then Harpertown processor. It is noteworthy that performance for larger models is more sensitive to the memory path characteristics of a particular processor and even more significant is the improvement with more number of threads. Most interesting is the performance comparison for a fully populated node. With 8 threads, Harpertown processor certainly hits a road block in memory path limit however the Nehalem processor performance is unaffected.

2.2 Factors influencing parallel scalability

It is well known that interconnect parameters (e.g. latency and bandwidth) are critical to the efficiency and scalability of complex cluster systems. Interconnect is critical for handling all communications requirements, transferring information from node to node without imposing a heavy overhead. Several publications have demonstrated the needs for a superior interconnect. It has been demonstrated that the low-bandwidth and high-latency characteristics of Gigabit Ethernet (GigE) make it a performance bottleneck, limiting the overall cluster efficiency and preventing any performance gain beyond four or five compute nodes. At the same time, an InfiniBand (IB) connected cluster has demonstrated higher performance (rating) than the GigE-based configuration [3]. Presently, Infiniband architecture provides for the most advanced interconnect for I/O and inter-process communications. High performance MPI implementations over Infiniband are also available. While a notable feature of Infiniband is its high bandwidth, the network bandwidth can still prove to be the performance bottleneck in some of today's most demanding applications. This is especially the case for clusters built with SMP nodes, in which multiple processes may run on a single node and must share a node bandwidth.

2.2.1 Host Channel Adapters (HCAs): Two ways to overcome bandwidth limitations are the use of more sophisticated Host Channel Adapter (HCA) cards and multi-rail networks. In this section, we highlight the importance of Host Channel Adapter (HCA) cards for servers and clusters with multi-core processors. A case study using FLUENT CFD code is performed on 2 pairs of node. The first pair of nodes is using IB HCA InfiniHost3 while the second pair of nodes is with IB ConnectX. The round robin allocation scheme is used so if we run on 2 cores then thread 0 is allocated on node 0, thread 1 is allocated on node 1. If we run with 4 threads, then threads 0 and 2 are allocated on node 0 and threads 1 and 3 get allocated on node 1 and so on. We intentionally chose a very small case (32,000 cells) so cost of communications will be quite high. Following are the timings from running the solver for 100 steps:

Table 3: Comparison of HCAs

# of cores (threads)	InfiniHost3	ConnectX	Speedup
2	9.605	8.214	1.17
4	5.687	4.351	1.31
8	3.368	2.560	1.32
16	6.504	1.812	3.59

It is obvious that IB ConnectX provides much better performance on a high number of threads. It is also noteworthy that it maybe much cheaper to upgrade only the HCA cards on a cluster then rewrite the code of interest.

2.2.2 Multi-rail Network: Another interesting and useful feature of multi-core Infiniband clusters is the multi-rail network. Multi-rail networks can improve MPI (Message Passing) communication performance by distributing the communication traffic to multiple independent networks (rails). For multi-rail, the MPI library has to manage the network traffic between the available separate network fabrics. The primary intent of having multiple network on a system is to separate communication and IO activities. However, several applications have limited IO activity and their performance may be limited by communication costs; so the choice of multiple networks seems very attractive towards improving communication costs in such application codes. One good example is the CFD steady state solvers. The main task of a steady state CFD solver is to arrive at a converged solution in the shortest possible time. Intermediate results are not of much interest so one will not necessarily save those results. With steady state codes, the primary IO activity typically happens in the initial and final phases of simulation so the main part of simulation time does not involve much IO activity.

The FLUENT standard benchmark suite, using FLUENT v12.0.9, running on an Altix ICE cluster is used to understand multi-rail network contribution towards improving the application performance. The cluster configuration is a 256 node Altix ICE+ (8200EX) cluster with 3.0GHz/12M/1600MHz 120W quad-core/node (x5472 Harpertown/Seaburg chipset) and 2GB/core memory. And, the performance metric is the same rating number used by ANSYS that represents the number of jobs completed in a 24-hour time frame. Hence, a higher number represents better performance in Table 4.

For smaller models, in general a single-rail network provides for sufficient bandwidth and there is little improvement in parallel scaling with dual-rail network. However, for larger cases, we see a significant difference between single-rail and dual-rail performance numbers beyond 512 cores. With a single-rail network, communication cost after 256 cores becomes expensive and we start to see negative scaling. At the same time dual-rail network has enough bandwidth capacity to sustain a reasonable growth of parallel performance expansive. Indeed, it is worth using multi-rail features for large scale simulations and at high core counts.

Table 4: Single-rail/Dual-rail performance rating comparison

nprocs	Turbo (0.5M cells)	Eddy (417K)	Aircraft (2M)	Sedan (4M)	Truck (14M)	Truck_poly (14M)
16	3680/3790	1134/1130	590/630	440/440	69/70	70/70
32	7513/7697	2160/2152	1209/1180	804/905	140/139	138/135
64	13880/14281	3516/3473	1691/2066	1788/1784	281/281	253/268
128	14706/14460	4867/4916	2551/4414	3435/3442	556/540	440/500
256	18885/19750	6484/6496	4179/5647	5477/5494	1009/1019	869/853
512		4028/6912	1837/2171	4661/7464	1006/1546	858/1842
1024			901/1353	692/862	803/2796	772/2374

2.2.3 Large Model Parallel Scalability*:* A large truck model of 111M cells of mixed type is used to understand the benefits of multi-rail network and appropriate tuning of the hardware features. The problem involves external flow over the truck body and uses DES model with the segregated implicit solver. The results that are presented below are using FLUENT Version 12.0.9 on Altix ICE 8200EX (3GHz, quad-core) can also be seen at the ANSYS benchmark website at: http://www.fluent.com/software/fluent/fl6bench/fl6bench_6.4.x/problems/truck_111m.htm

Table 5: External flow over a truck, FLUENT v12.0.9, Altix ICE 8200EX scaling

CPUs of Cores	Machines or Blocks	Rating (higher number is better)
128	16	60
256	32	124.1
512	64	244
1024	128	464.3
2048	256	741

3.0 CONSIDERATIONS FOR FUTURE WORK

For parallel scalability a major contributing factor therefore is the performance of a particular MPI implementation on any particular computer architecture. A common belief is that the major part of parallel performance is defined by the quality of point-to-point communications. In general it is a valid statement but translating it to the network requirements can be non trivial. A simple way to state this is as follows: For a one level algorithm where all computations are done on one computational grid, the size of inter domain messages is defined by a size of domain faces and because it is typically quite large it is dependent on the network bandwidth. On the other hand for multi-level solvers (e.g. multi-grid algorithms) a large part of the communications is performed with small size messages and therefore depends mostly on network latency. However, such requirements will depend also on the problem size, number of nodes and other parameters.

In order to get an understanding of the communication scenario, performance instruments are wrapped around MPI calls that report the time spent in a specific communication primitive at runtime. Below is an example of MPI session timing taken from a CFD run on 32 cores (8 nodes):

MPI_Allreduce 53.1 sec; *MPI_Waitall* 13.2 sec; *MPI_Send* 7.2 sec; *MPI_Recv* 8.3 sec; *MPI_Barrier* 5.2 sec
MPI Time 112.3; *Wall Time* 374.4

Clearly, a large part of time is spent in the collective operator MPI_Allreduce. However, when we instrument the same example with a tool (MPIinside) that uses an explicit synchronization call just before MPI_Allreduce we observe the following: *MPIinside statistics*: b_Allred 53.3446 sec; Allred: 5.6589 sec

Here b_allred is the time to synchronize and allred is the time spent by the collective operator itself. As we can see the actual cost of collective operations is lower then point-to-point communication primitives but the implicit synchronization time which is required for completion of collective operations is an expensive part of the overall communication cost. This application has almost ideal load balance measured by number of cells per parallel domain so the computation time scales well and according to Amdahl's law the overall scaling should also be good. However, we can see that a code will wait in certain specific points for implicit synchronization which we will refer

to as micro-imbalance. The reasons for this imbalance can be several including that the actual amount of compute work is defined not only by number of computational cells but also through the variations of the physical models, variations of computational domains like internal and boundary domains, fluctuations in HCA loads, etc...The elimination of micro-imbalance can be a major factor in the future improvement of parallel scalability. One approach that can be utilized is the new MPI protocol (MPI-2) that introduces a new class of communication primitives called one-sided communications, example of such primitives being PUT and GET. Such an approach relies on large shared memory windows and minimal amount of direct copy operations which in turn leads to much better MPI performance. Using one-sided communications, one can observe the increase in bandwidth and even more impressive is the decrease in latency by a factor of about 3X. Another wonderful feature of one-sided communications is the asynchronous nature of communication algorithms based on those primitives.

The above performance enhancements in single discipline CFD codes allow engineers not only to perform more sophisticated and realistic fluid dynamic analyses but also permit the use of high fidelity CFD codes as an integral part of more complex design processes like Multidiscipline Analysis and Optimization (MDO) [4]. For example, a basic aircraft design does not require much computational power, however, in order to be more precise and model a more refined shape, 100's of design variables will be required to represent the variation in the shape of the airfoil at many stations along the wing as well as details of the internal structure. Clearly, this would call for a high fidelity CFD analysis of the air flow, corresponding to each change in design with the MDO process. The need for HPC is therefore ubiquitous for several reasons including: faster turn around times of solutions to complex design problems; dealing with ever-growing model sizes and more complete models such as in rotor-stator interactions in a turbine; handling more complex physics such as in combustors and Large Eddy Simulations; and, enabling formal MDO solutions in a timely manner to impact product design cycle times.

Key HPC requirements to systematically and rigorously investigate a large design space for MDO include:

- Balanced HPC environments to support multidiscipline analysis: The heterogeneous mix of simulations common to a MDO process tends to exhibit a range of HPC resource demands. For instance, the implicit structural analysis solvers for dynamic responses requires high rates of memory and IO bandwidth with processor speed as a secondary concern while explicit dynamics solvers for impact crash simulation benefits from a combination of fast processors for the required element force calculations and a high rate of memory bandwidth necessary for efficient contact resolution. CFD also requires a balance of memory bandwidth and fast processors, but benefits most from parallel scalability.
- High throughput efficiency: Most vehicle systems are multidisciplinary involving a heterogeneous mix of analysis codes, including high fidelity analyses codes; High throughput efficiency on a multiprocessor system allows for fast turnaround times of multiple jobs, enabling many runs to be made concurrently in a short amount of time as required in the construction of the approximation (surrogate) models for MDO [5].

For today's economics, these HPC resources such as CPU cycles, memory, system bandwidth and scalability, storage, file and data management – must deliver the highest levels of engineering productivity and HPC reliability that is possible from a platform environment.

4.0 SUMMARY

Parallel processing on high performance computing clusters is enabling much larger and more complex CFD problems. However, in order to realize these performance gains, a knowledgeable use of the latest generation of HPC clusters with multi-core processors, multi-rail networks, MPI and such developments is critical. This paper details possible ways to improve CFD code performance on HPC clusters and the standard benchmark suite of the industry standard CFD code, FLUENT, is used in illustrating the performance improvements. Further, this paper addresses the ubiquitous need for HPC with high fidelity, multidisciplinary analysis and optimization.

REFERENCES

1. A. Jameson and M. Fatica, "Using Computational Fluid Dynamics for Aerodynamics," Stanford University, Stanford, California, 2006.
2. S. Kodiyalam, M. Kremenetsky and S. Posey, "Balanced HPC Infrastructure for CFD and associated Multidiscipline Simulations," Proceedings, 7th ASIAN CFD Conference, Bangalore, India, Nov 2007.
3. G. Shainer, S. Kher, & P. Alavilli, "Interconnect Helps Efficiency in Clusters," Desktop Engineering, Dec 2008.
4. J. S. Sobieski and R. T. Haftka, "Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments," Structural Optimization, pp. 1-23, Vol. 14, No. 1, August 1997.
5. S. Kodiyalam and J. S. Sobieski, "Bi-Level Integrated System Synthesis with Response Surfaces," AIAA Journal, Vol. 38, No. 8, pp. 1479-1485, August 2000.

Efficient Parallel Algorithm for Aerodynamic Design of Wing-Body-Junction Driven by Accurate Navier-Stokes Computations

Sergey Peigin*, Boris Epstein**

**Israel Aerospace Industries, Lod, 71700, Israe*
(e-mail: speiginr@iai.co.il)
***The Academic College of Tel-Aviv-Yaffo, Tel-Aviv, Israel*
(e-mail: epstein@mta.ac.il)

Abstract: A CFD driven robust and accurate method for constrained aerodynamic design previously developed by the authors, is extended to complex aircraft junctions such as wing-body-fairing.In this method, the total drag is minimized at fixed lift subject to numerous geometrical and aerodynamical constraints. The method is driven by Genetic Algorithms and full Navier-Stokes computations supported by massive multilevel parallelization. The method is illustrated by example of the optimization of wing-body-fairing for generic business jet configuration at realistic transonic cruise flight conditions. The results indicate the applicability of the method to practical aerodynamic design.
Keywords: Aerodynamic design, drag reduction, efficient parallelization strategy

1. INTRODUCTION

It is generally expected in aircraft industry that automatic or even semi-automatic optimizers may essentially shorten the process of aerodynamic design, especially at the preliminary design stage, which may cost above 100 million USD. Alongside the reduction in the design costs, accurate optimizers may improve the quality of design making the project more competitive.

The authors' opinion is that the following reasons may contribute to the insufficient use of 3D CFD driven optimization tools for automatic aerodynamic design. Firstly, most of these methods employ incomplete gas-dynamic models. E.g. the use of inviscid Euler equations as a CFD driver, limits the accuracy of optimization where viscous effects are significant. Secondly, wide-spread gradient-based optimization methods are confined to the search of local extrema which inevitably affects the globality of optimization. The third, and probably most fundamental drawback of the existing methods is that they are able of handling only a very limited (if any) number of constraints, while the industrial optimization requires a significant (about 20-30) number of aerodynamic and geometrical constraints placed upon an optimal shape.

With the purpose to overcome these difficulties, in this paper an accurate, robust and computationally efficient approach to the constrained optimization of essentially 3D non-linear surfaces typical of complex aircraft junctions such as wing-body-fairing is presented.It is based on the complete aerodynamic model (Navier-Stokes computations), expands the globality of optimum search (by employing mixed deterministic/probabilistic Genetic Algorithms), and handles simultaneously a large number of constraints. The present algorithm essentially extends the capabilities of the optimization tool OPTIMAS (previously developed by the authors [1-3] to a significantly higher level of geometrical complexity of optimised aerodynamic configurations.

In industry, the success of the method should be assessed through practical optimization of realistic aerodynamic configurations. In this paper, we illustrate the capabilities of the method by applying it to the optimization of wing-body-fairing for a generic business jet configuration at realistic transonic cruise flight conditions.The results indicate the applicability of the method to practical aerodynamic design in engineering environment. It was shown that the optimization of wing-body-junction can considerably improve the aerodynamic performance of the aircraft in comparison to the previous design where only wing surface of the generic business jet was subject to optimization.

2. OPTIMIZATION ALGORITHM

As a basic search algorithm, a variant of the floating-point GA [4] is used.The mating pool is formed through the use of tournament selection.This allows for an essential increase in the diversity of the parents. We employ the arithmetical crossovercand the non-uniform real-coded mutationcdefined by Michalewicz [4]. To avoid a premature convergence of GA we applied the mutation operator in a distance-dependent form. To improve the convergence of the algorithm we also use the elitism principle.

The constraints handling in the framework of GAs search was based on the following approach [5]. Contrary to the traditional approach, we employ search paths which pass through both feasible and infeasible points. To implement this we extend the search space to the infeasible region using the important property of GAs*:* contrary to classical optimization methods, GAs are not confined to only smooth extensions.

It is assumed that the geometry of the aircraft configuration is described in the absolute Cartesian coordinate system *(x,y,z)*, where the axes *x,y* and *z* are directed along the streamwise, normal to wing surface and span directions, respectively. In the developed approach, the whole surface of a wing body configuration is divided into three parts. The first part contains the points of the aircraft fuselage ``inboard of the fairing.''This part of the configuration is not subject to modification.

The second part contains the points of the exposed wing ``outboard of the fairing''. This part of the configuration is represented by a linear interpolation of 2D cuts (wing sections). For each wing section, the non-dimensional shape of the airfoil (scaled by the corresponding chord) is defined in a local Cartesian coordinate system. For approximation of the upper and lower airfoil surface, Bezier curve of order *N* (one-dimensional Bezier Spline) representation was used.

Finally, the third (highly non-linear) part of the configuration is the fairing. This essentially 3D part is described through combination of Bezier surfaces representation (two-parameter families of Bezier Splines of order *N* and *M*) and local twist distribution (one-parameter Bezier Spline of order *N*).

Finally, the dimensions N_D of the search space are equal to*:* $N_D=(2N-2)(M-1)$. In practice, we used $N=10$, $M=4$ making $N_D=54$.

One of the main weaknesses of GAs lies in their low computational efficiency. That means that special efforts should be made in order to significantly improve the computational efficiency of the algorithm. This was accomplished through the application of the following two approaches.

In the framework of the first approach, we use Reduced-Order Models theory in the form of Local Approximation Method (LAM). The idea is to approximate the cost function using information from the specially constructed local data base.The data base is obtained by solving the full Navier-Stokes equations in a discrete neighbourhood of a basic point positioned in the search space. Specifically a mixed linear-quadratic approximation is employed. One-dimensionally, the one-sided linear approximation is used in the case of monotonic behaviour of the approximated function, and the quadratic approximation is used otherwise.

Besides, in order to enhance the global character of the search, we perform iterations in such a way that in each iteration, the result of optimization serves as the initial point for the next iteration step (further referred to as optimization step).

In the framework of the second approach we use an embedded multilevel parallelization strategy [6] which includes Level 1 - Parallelization of full Navier-Stokes solver*;* Level 2 - Parallel CFD scanning of the search space*;* Level 3 - Parallelization of the GAs optimization process*;* Level 4 - Parallel optimal search on multiple search domains.

Finally we can conclude that the multilevel parallelization approach allowed us to sustain a high level of parallel efficiency on massively parallel machines, and thus to dramatically improve the computational efficiency of the optimization algorithm.

3. ANALYSIS OF RESULTS

The method was applied to the problem of a single point multiconstrained optimization of the wing-body junction surface for a generic business jet aircraft at transonic flight conditions. In total 4 test cases were considered. The corresponding optimal shapes are designated by *Case_GBJFR_1* to *Case_GBJFR_4*.

For the subsequent discussion we will consider the following three initial geometries of the generic business jet: the original geometry and two geometries with already optimized exposed wing, which were presented in [3] and labelled in this paper as *Case_GBJ_2* (no constraint on C_M) and *Case_GBJ_5* (with constraint on pitching moment).

The test case *Case_GBJFR_1* deals with fairing optimization (with frozen exposed wing) without constraint on pitching moment and without beam constraints. The initial geometry for this optimization came from *Case_GBJ_2*. The next test case (*Case_GBJFR_2*) deals with wing optimization with the frozen wing-body fairing resulted from the *Case_GBJFR_1* optimization.

The last two test cases *Case_GBJFR_3* and *Case_GBJFR_4* correspond to optimization with constraint on C_M and with beam constraints, which were kept to the original level. For *Case_GBJFR_3* (fairing optimization with frozen optimal wing) the initial geometry was *Case_GBJ_5*, while for *Case_GBJFR_4 (*exposed wing optimization with frozen optimal fairing) the initial geometry was the optimal shape of *Case_GBJFR_3*.

For the original configuration, the combination of flight conditions at the main cruise design point(a high free-stream Mach number M=0.80 and a moderate transonic lift coefficient C_L=0.40*)* leads to the development of a lambda-like spanwise shock development. At these conditions the drag value for the original configuration is equal to C_D=292.0 aerodynamic counts.

The previously performed in [3] unconstrained one-point wing optimization with the frozen fairing shape (*Case_GBJ_2*) allowed to achieve 16.7 counts of drag reduction and decrease the total drag down to 275.3 counts. More detailed analysis shows that this significant drag reduction may be attributed to a significant decrease in the shock strength in the exposed wing region. Nevertheless, the shock intensity at the wing-body-junction region was not practically diminish.

The optimization of the wing-body-fairing shape (*Case_GBJFR_1*) improved the pressure distribution in the fillet region. As a result, this improvement permits to reduce the total drag of the optimized configuration by an additional 10.7 counts (compare to *Case_GBJ_2*) and to achieve the level C_D=264.6 counts.

The second test case *Case_GBJFR_2*, starting from *Case_GBJFR_1* as an initial geometry, deals with the optimization of the exposed wing keeping the fairing shape frozen. The optimization *Case_GBJFR_2* further improved the aerodynamic performance of the aircraft configuration: the total drag value is equal to 258.7 drag counts (5.9 counts less than that of the initial geometry *Case_GBJFR_1*).

Off-design behaviour of the optimized configurations may be studied through lift/drag polars. It can be concluded that the local gains described above are preserved in a wide range of lift coefficient values from C_L=0.15 to above C_L=0.6. Additionally we should note that the drag reduction due to optimisation increased for Mach numbers higher than the design value. Specifically, at M=0.82 and C_L=0.40 the drag reduction of *Case_GBJFR_2* with respect to the original business jet configuration, is equal to 42 counts.

Alongside the unconstrained (with respect to C_M and local thickness) optimisations, the optimizations with beam constraints and with the constraint on pitching moment, were performed. In *Case_GBJFR_3* and *Case_GBJFR_4*, the value the corresponding constraints were kept to the original level.

For *Case_GBJFR_3* the initial geometry was the optimal shape of the one-point constrained drag minimization of the exposed wing with fixed fuselage and fillet shapes (labeled as *Case_GBJ_5* in [3]). The total drag value for *Case_GBJ_5* configuration was equal to *276.1* counts, while the fairing optimization reached the level of 269.2 drag counts (in total about 7 counts drag reduction).

Let us now consider the results achieved in the last test case *Case_GBJFR_4*. Starting from the *Case_GBJFR_3* aircraft shape as an initial geometry, this constrained optimization deals with the modification of the exposed wing keeping the wing-body fillet shape frozen.

The corresponding drag reduction was equal to 10.1 counts. It is important to note that the achieved drag level (C_D=259.1 c, C_M=-0.136) is only a little bit higher in comparison with *Case_GBJFR_2* (C_D=258.7 c, C_M=-0.149). This means that, in the framework of the successive optimisations of the exposed wing and of the fillet region, the penalty due to the imposition of the pitching moment constraint and beam constraints is negligibly small (about 0.4 counts). Moreover, at a higher Mach number M=0.82, the drag reduction for *Case_GBJFR_4* compared to the original aircraft configuration was equal to 42.7 counts (from the original *321* counts down to 278.8 counts), while the total drag value for *Case_GBJFR_2* was equal to 280.8 aerodynamic counts.

Finally, let us present data which illustrate the off-design behaviour of the optimal geometry of *Case_GBJFR_4*: lift/drag polars at M=0.80 and M=0.82 and Mach drag rise curves at the design C_L=0.4. As a whole the analysis shows that the considered optimisation is not restricted to a single design point and that the gain due to optimization is observed in a wide region of lift values and free-stream Mach numbers. For example, the optimization succeeded in shifting the Mach drag divergence point to at least the main Mach design value. Specifically, based on the definition of the Mach drag divergence point, the corresponding M_{DD} value for the original configuration is equal to 0.795, while for *Case_GBJFR_2* and *Case_GBJFR_4* M_{DD} =0.815. Additionally, the subsonic drag level of the optimized configuration is also decreased (apparently due to reduction in the value of form drag).

REFERENCES

1. Epstein, B., and Peigin, S, (2004). Robust Hybrid Approach to Multiobjective Constrained Optimization in Aerodynamics. *AIAA Journal*,} Vol. 42, No.8, pp. 1572--1581.
2. Peigin, S., and Epstein, B, (2006). Computational Fluid Dynamics Driven Optimization of Blended Wing Body Aircraft. *AIAA Journal*, Vol. 44, No. 11, pp.2736-2745.
3. Epstein, B., and Peigin, S ,. (2007). Efficient Approach for Multipoint Aerodynamic Wing Design of Business Jet Aircraft. *AIAA Journal*, Vol. 45, No. 11, pp.2612-2621.
4. Michalewicz, Z, (1996). *Genetic Algorithms + Data Structures = Evolution Programs* ,Springer Verlag, New-York.
5. Peigin, S., and Epstein, B, (2004). Robust Handling of Non-linear Constraints for GA Optimization of Aerodynamic Shapes. *Int. J. Numer. Meth. Fluids*, Vol. 45, No. 8, pp.1339—1362
6. Peigin, S., and Epstein, B, (2004). Embedded Parallelization Approach for Optimization in Aerodynamic Design. *The Journal of Supercomputing*, Vol. 29, No. 3, pp.243--263 .

Adjoint-Based Adaptive Meshing and Shape Optimization in a Parallel Setting

Marshall Gusman*, Jeffrey Housman, Cetin Kiris*****

*U.C. Davis & Eloret Corp., NASA Ames Research Center, Moffett Field, CA 94035, USA
(Tel: 650-604-5455; e-mail: mrgusman@gmail.com)

**U.C. Davis & Eloret Corp., NASA Ames Research Center, Moffett Field, CA 94035, USA
(Tel: 650-604-5455; e-mail: Jeffrey.A.Housman@nasa.gov)

*** NASA Ames Research Center, Moffett Field, CA 94035, USA
(Tel: 650-604-4485; e-mail: Cetin.C.Kiris@nasa.gov)

Abstract: Three families of axis-symmetric fairing shapes are studied in subsonic through supersonic flow conditions to assess drag performance over the ascent trajectory of the Saturn V launch vehicle. To quantify and compare the drag performance of the various fairing shapes, a quantity known as the 'drag loss' is introduced. Solving the adjoint equations, or dual problem, enables the calculation of error bounds and the creation of intelligently focused mesh refinement for a highly efficient parallel solver. The results reported in this paper were computed with Cart3D on the Columbia supercomputer located at NASA Ames Research Center.

Keywords: adjoint, fairing, shape optimization

1. INTRODUCTION

During the Apollo program of the 1960's and 1970's, NASA's Saturn V launch vehicle successfully launched many orbital and lunar missions. This paper describes the application of Cart3D, a highly parallel inviscid CFD tool with adjoint-based mesh refinement and shape optimization capabilities, applied to the Saturn V rocket for the purpose of predicting pressure drag on the fairing. Because boundary layer growth is negligible over the relatively short length of the fairing, and skin friction effects will be minimal, the inviscid assumption will still provide accurate comparisons of drag performance. Parallel computations with an adaptive-meshing Eulerian flow solver provide fast, efficient, and accurate predictions of drag on a rocket fairing.

The application of advanced CFD tools such as Cart3D to the design and analysis of complicated geometry has become increasingly affordable, and rapid and accurate prediction of aerodynamic forces is now possible. For example in this study, subsonic and transonic flow simulations of the Saturn V geometry are completed in less than one hour using 24 processors on Columbia, and less than 30 minutes for supersonic cases. The parallel capability of the code enables the creation of large databases of flow solutions with reasonable turnaround time. This ability to analyze many shapes throughout an ascent trajectory makes parallel CFD an important tool in the design process.

This paper begins with an introduction to the Cart3D flow-solver package. In particular, a discussion of the new adjoint-based adaptive meshing capabilities and shape optimization, as described in Nemec and Aftosmis [3] and [4]. Next, a review of the original Saturn V launch vehicle is presented, along with a description of the parameters and processes considered in this study. Simulation results for the various fairing shapes are then presented with an analysis of the drag loss.

With the impending retirement of the Space Shuttle, NASA has renewed interest in traditional rocket designs for its next generation launch vehicles. This academic study of the Saturn V is also applicable to the design and analysis of NASA's next generation launch vehicles, but is not published here.

1.1 Study Description

The objective of the fairing design study is to identify a shape with the least drag over the Saturn V's Stage I ascent trajectory. A subset of points from the ascent trajectory [5] is chosen and steady-state runs at each of these points are

computed. The Mach numbers included in this study range from 0.6 to 5.0. Three families of fairing shapes are examined, comprising: conical with round tip, Sears-Haack (2 shapes), and power-law (3 shapes), seen in Figure 1 mounted on a simplified Saturn V geometry. The dimensions of the original Apollo command module define the total length L = 113.34 inches, measured from the tip to the cylindrical stack, and maximum radius R = 77.00 inches.

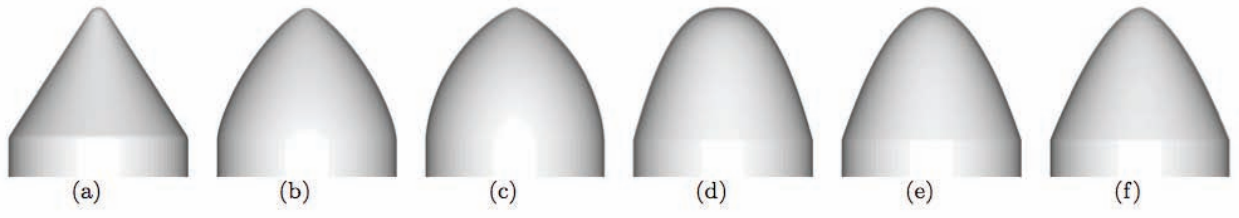


Figure 1: Shapes included in study (a) Baseline, (b) Sears-Haack C=0.00, (c) Sears-Haack C=0.33, (d) Power-Law N=0.4, (e) Power-Law N=0.5, (f) Power-Law N=0.6

1.2 Flow Solver Description

The inviscid flow-solver Cart3D, see Aftosmis et al. [1], is used to quickly and efficiently solve a test matrix of geometric configurations and flight conditions using an adaptively refined Cartesian cut-cell approach. This cut-cell approach decouples the volume grid from the surface representation, allowing adaptive grid generation without the need to regenerate the surface mesh (and project back to CAD). The solution is marched in time to steady-state convergence using a five-stage Runge-Kutta scheme with local time stepping, and a multigrid W-cycle to improve solution convergence. Domain decomposition allows parallel processors to quickly solve any given problem with nearly linear speed-up.

1.3 Adjoint Adaptive Meshing

In order to accurately predict the aerodynamic functionals of interest, an adjoint-based adaptive mesh refinement scheme is implemented in Cart3D, see Nemec and Aftosmis [4]. This method uses the solution of the adjoint equations to produce a local error estimate for each cell, a criteria which is used to identify cells for further refinement. The result is a reduction in the local error contribution of the cells in error prone-regions, and an overall improved estimate for the functional of interest. This procedure generates an efficient mesh for computing the functional by refining only necessary cells and minimizing computational time.

Central to the proper use of the methodology is the concept of the functional (J (Q)). The functional is a scalar quantity that depends on the flow solution, such as the drag or lift of an aerodynamic body. A discrete mesh can only support a functional approximation of limited-accuracy, but better accuracy can be obtained by 'embedding' the mesh with universal refinement. The error between these functional approximations is determined by the following procedure, which is then used to identify flow regions for mesh refinement.

First, the functional error $|J(Q_h)-J(Q_H)|$, on the initial mesh must be approximated, where the subscript Q_h designates an embedded mesh solution and Q_H refers to the initial coarse mesh. Using a Taylor series for the approximation of the functional and residual on the fine mesh, we can calculate the error in the functional on the coarse mesh

$$Functional\ Error = |J(Q_h) - J(Q_H)| \approx |\psi_h^T R(Q_h)| \tag{1}$$

where ψ_h is the solution to the adjoint equation

$$\left[\frac{\partial R(Q_h)}{\partial Q_h}\right]^T \psi_h = \frac{\partial J(Q_h)}{\partial Q_h}^T \tag{2}$$

The linear system of equations defined by (2) is related to the discrete flow equations through a duality principle, see Giles and Pierce [2]. With knowledge of the local error contributions of each cell, a new mesh is created with better refinement in error-prone regions. The cells with error above a certain user-defined tolerance are refined, while the cells that are below the tolerance are maintained at their previous resolution level. This process is repeated until a user-defined 'global error tolerance' is achieved or the maximum number of adaptation levels has been generated.

Figure 2 shows slices of the mesh at successive refinements during the solution process for the baseline shape in supersonic flow. For this study, refinement is limited to the regions affecting drag on the fairing (the functional). Mesh points cluster at the shock and near the body where the highest errors exist, while regions downstream of the fairing are ignored. This localized refinement demonstrates the effectiveness of the adjoint method, as only the influential regions are refined. The same parameters are specified for the geometries in the shape study, and similar high-quality meshes are obtained.

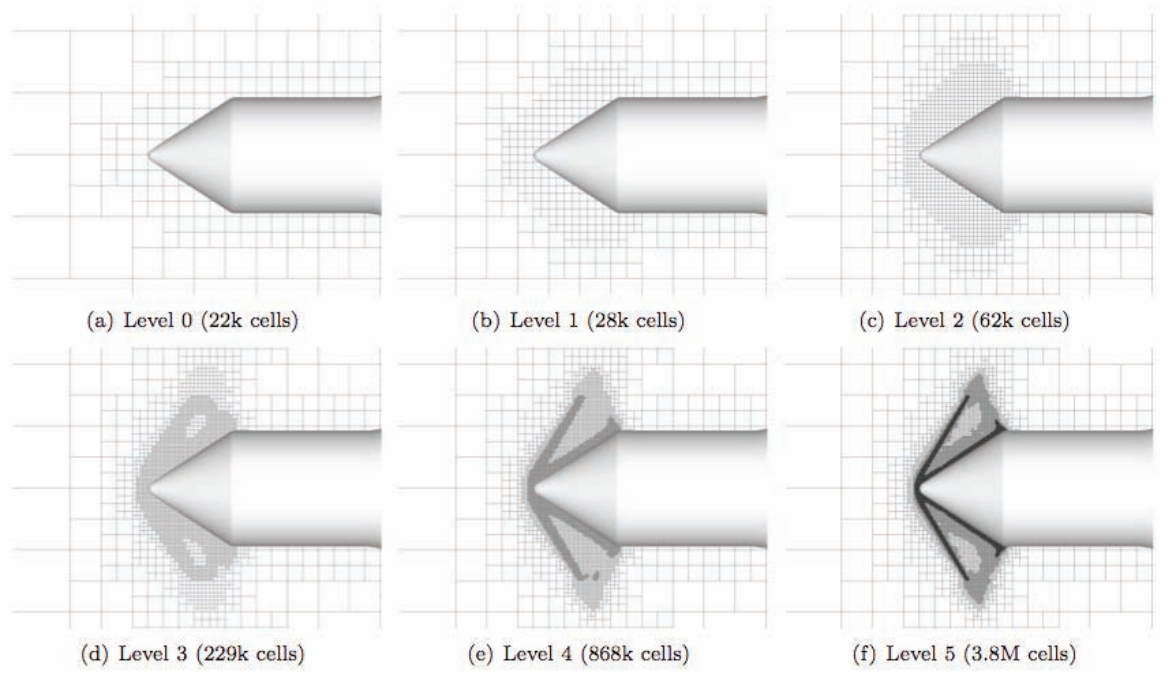


Figure 2: Mesh adaptation on baseline Saturn V shape, Mach 1.70

2. RESULTS

A brief verification of the adaptive meshing scheme is presented here. For the baseline shape at Mach 1.7, Figure 3 shows the drag coefficient as a function of the mesh size. Initially the 22,000-cell mesh under-predicts C_D by 7% in comparison to the final 3,800,000-cell mesh. The corrected functional is also shown, which is calculated from the correction term in the adjoint equation. On a given mesh, the corrected functional is an estimate of the functional value on a universally refined embedded mesh, corresponding to the next-level of mesh adaptation. Both values converge towards the asymptote where $C_D \approx 0.805$.

Progressive improvements to the functional accuracy are observed in Figure 4, where the functional error is plotted against mesh size. From the first adapted mesh of approximately 27,000 cells, the log of error falls linearly with the log of the number of cells. This behavior is extrapolated to the final mesh where error is approximately 0.003, or $<$ 0.5%. The adjoint solution is not computed on the final mesh because no further mesh refinement is performed, however, it could be computed to confirm the maximum error and to provide an even better estimate of the functional.

Final results of the fairing study indicate a preference for blunt shapes, due to their superior performance in the transonic flight regime. Figure 5 shows a comparison of the drag coefficients for the power-law and Sears-Haack families of shapes across the entire ascent trajectory. Fairings are qualitatively compared based on a weighted integral of the drag over flight time (drag loss). A full discussion of the results and explanations of the adjoint-based mesh adaptation and shape optimization are presented in the full paper. Parallel computations are shown to be an integral component of modern aerodynamic shape analysis and design.

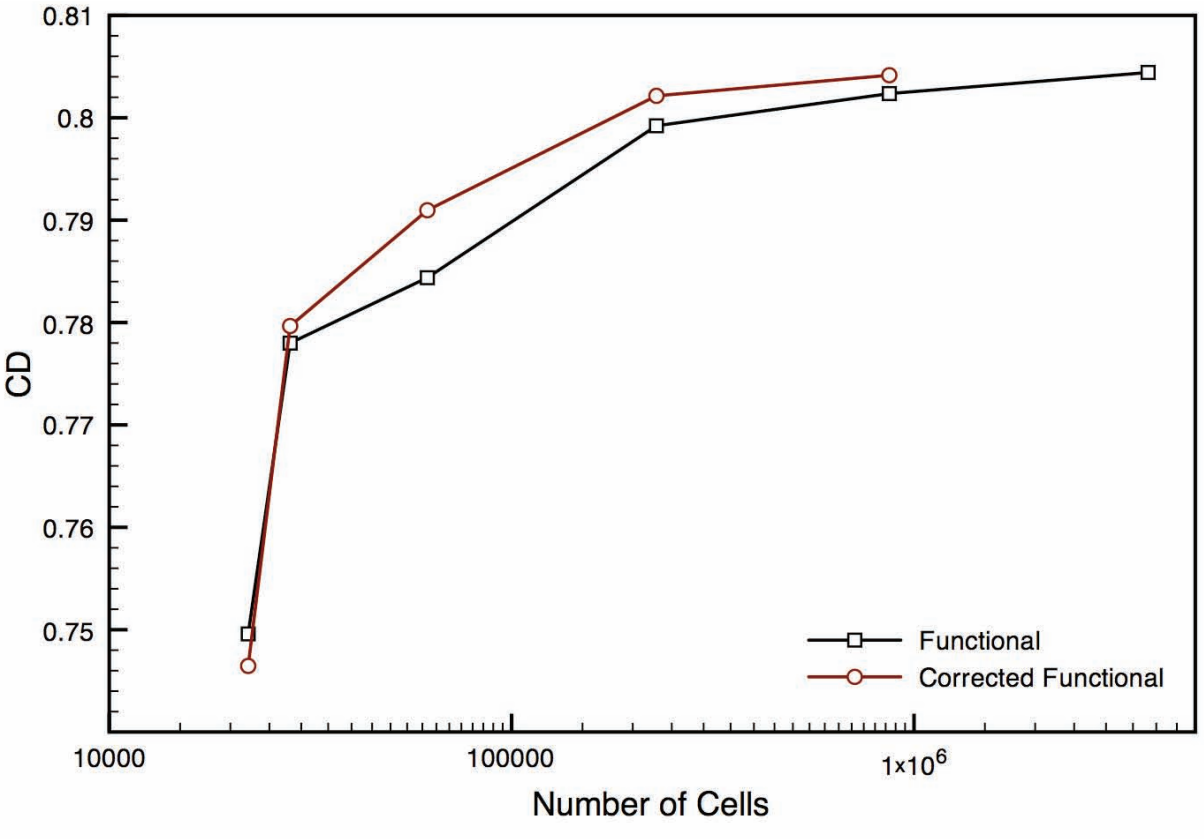


Figure 3: Drag coefficient convergence for baseline shape at Mach 1.70

ACKNOWLEDGEMENTS

We thank Marian Nemec and Mike Aftosmis for their impeccable Cart3D support.

REFERENCES

[1] Michael J. Aftosmis, M. J. Berger, and G. Adomovicius. A parallel multilevel method for adaptively refined cartesian grids with embedded boundaries. AIAA, (2000-0808), 2000.

[2] M. B. Giles and N. A. Pierce. Adjoint equations in CFD: duality, boundary conditions and solution behavior. AIAA, (97-1850), 1997.

[3] Marian Nemec and Michael J. Aftosmis. Aerodynamic shape optimization using a cartesian adjoint method and CAD geometry. AIAA, (2006-3456), 2006.

[4] Marian Nemec and Michael J. Aftosmis. Adjoint error estimation and adaptive refinement for embedded-boundary cartesian meshes. *In 18th AIAA Computational Fluid Dynamics Conference*, AIAA-2007-4187, 2007.

[5] Saturn V flight evaluation working group. Saturn V launch vehicle flight evaluation report-AS-503 Apollo 8 mission, February 1969.

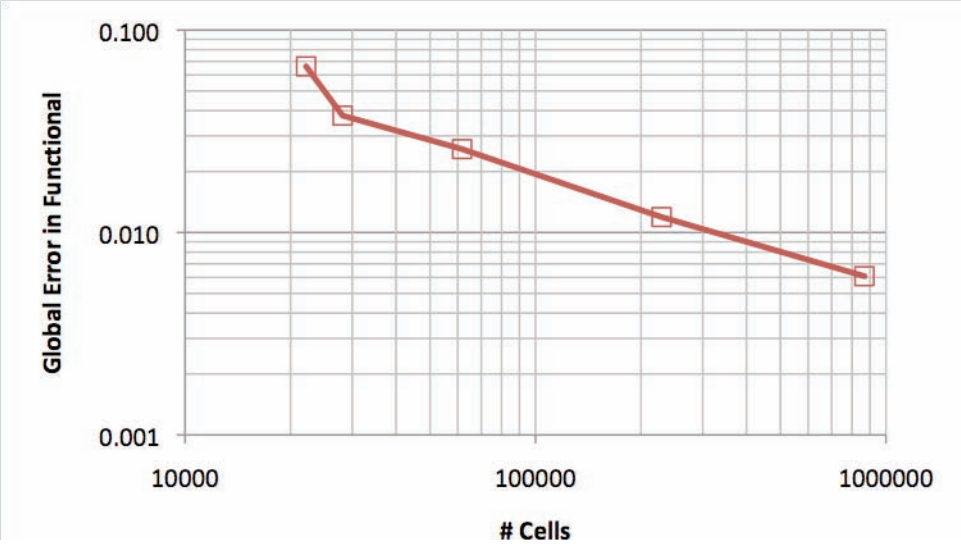


Figure 4: Global error reduction, baseline shape at Mach 1.70

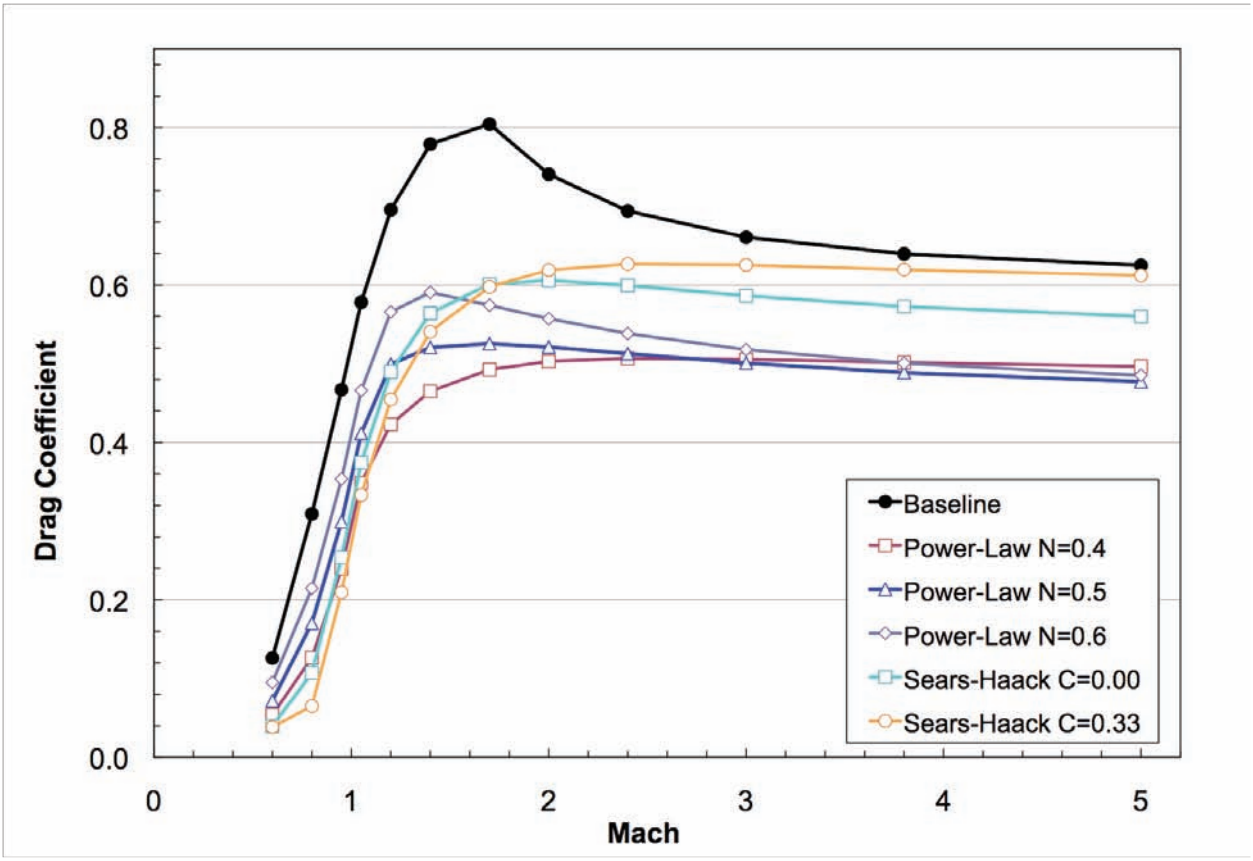


Figure 5: Drag coefficient of fairing shapes over launch trajectory

Parametric Co-Optimization of Lifting Blunt Body Vehicle Concepts for Atmospheric Entry

Joseph A. Garcia*, James L. Brown, David J. Kinney**, Jeffrey V. Bowles**, Loc Huynh*****

*NASA Ames Research Center, Moffett Field, CA 94035, USA
(Tel: 650-604-0614; e-mail: Joseph.A.Garcia@nasa.gov)
** NASA Ames Research Center, Moffett Field, CA 94035, USA
*** ELORET, Sunnyvale, CA 94087, USA

Abstract: An integrated Multi-disciplinary Design Optimization (MDO) technique for the development of planetary atmospheric entry vehicle concepts accounting for shape, trajectory, thermal protection system (TPS), and vehicle closure is described. This MDO technique aims to provide a practical approach to exploring new vehicle concepts to meet the increased demand of future space exploration mission requirements. This has been accomplished using a combination of engineering and higher-fidelity physics analysis tools along with optimization methods and engineering judgment. This integrated MDO process allows engineers to efficiently and consistently analyze multiple design architectures. The MDO environment is created using the ModelCenter software developed by Phoenix Integration. The framework described is referred to as COBRA, a rough acronym for "Co-Optimization of Bluntbody Re-entry Analysis". The COBRA framework utilizes parallel computing capabilities in two forms. One form is the use of parallel discipline codes, such as the aerothermodynamics analysis tools. Secondly, multiple design cases can be run simultaneously through the MDO framework. This parallelization approach allows the evaluation of complicated integrated systems with the proper level of physics within a reasonable turnaround time to impact and influence early design decisions.

Keywords: Multi-disciplinary, Re-entry, Planetary, Trajectory, Thermal, Protection

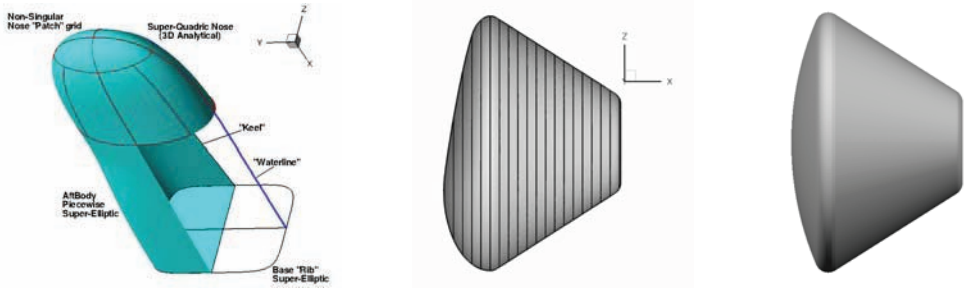
1. INTRODUCTION

In this paper we describe an integrated Multi-disciplinary Design Optimization (MDO) technique for the development of planetary atmospheric entry vehicle concepts accounting for shape, trajectory, thermal protection system (TPS), and vehicle closure. The framework described is referred to as COBRA, a rough acronym for "Co-Optimization of Bluntbody Re-entry Analysis". The paper will include details of the framework, the wrapped engineering and high-fidelity physics based analysis codes, the set of parameters being optimized, and the system constraints that have significant impact on the design of future atmospheric entry vehicles. We also describe preliminary results of such a design.

2. DISCIPLINARY ANALYSIS TOOLSET

2.1 Geometry

The vehicle geometry is based on one of several FORTRAN codes written specifically to provide an analytic description of the vehicle shape with a small number of geometric parameters. Examples shapes are shown in Fig 1.



Cobra ACVe [Ref. 1] Apollo/CEV

Figure 1: Example shapes

These codes enable the use of a small set of geometric parameters that define the vehicle's outer mode line (OML) geometry as part of the MDO algorithmic process providing shapes with a range of aero/aerothermodynamic properties useful for optimization. The general parametric shapes of the COBRA process entirely defines the vehicle shape without the need for intense human interaction, allowing for automated optimization of a large design space to find the best combination of aerodynamic stability and aeroheating for the vehicle performance.

2.2 Aerodynamics/Aerothermodynamics

The aerodynamic and aerothermodynamic characteristics of each particular vehicle shape is generated by either the CBAERO engineering code [2] and or DPLR [3] CFD code. CBAERO is an engineering analysis code based on independent panel methods, such as modified Newtonian, along with a surface streamline algorithm, and an extensive set of validated engineering correlations to establish surface pressure, convective and radiative heating, shear stress, and boundary layer properties. DPLR is a high-fidelity physics-based real-gas Navier-Stokes code used to give results either in support of or in place of CBAERO.

It is in this discipline where a hybrid approach is used which leverages high fidelity analyses with engineering methods using sophisticated interpolation techniques known as "anchoring". The anchoring approach aims to address the deficiency in the engineering and high fidelity methods, providing a rapid and intelligent engineering-based interpolation method. Further detail of this anchoring approach can be found in Reference [4].

2.3 Trajectory

During optimization of the vehicle, the nominal design trajectory must be considered to be dependent on the aerodynamic and aeroheating properties of the particular vehicle shape, in particular that of L/D, Ballistic Coefficient and peak heating. Constraints on the trajectory flight dynamics also must be imposed. To find the nominal trajectory for each vehicle under consideration, the POST2 [5] trajectory code is used.

2.4 Structures

Estimates of the impact of the vehicle shape on its structural weight are being planned through the use of NASTRAN [6]. This will allow us an understanding of how a shape that might be favorable for aerodynamics and TPS weight may prove to be unfavorable for structural efficiency. This will open up an additional useful trade space.

2.5 Thermal Protection System (TPS) Sizing

To assess the thermal protection system for the vehicle shape generated, the nominal trajectory for each such shape, and its aerodynamic and aeroheating characteristics, including the time history of the heating environments, are provided to the TPSSizer [7] set of programs, which includes the FIAT [8] thermal analysis code for ablative TPS materials. The result is a sizing of the TPS for a vehicle shape undergoing its own nominal trajectory for the mission constraints being considered.

2.7 Vehicle Sizing

Another constraint in the MDO algorithm is the launch vehicle. Specifically, its payload fairing and delivered vehicle total mass at entry interface for the atmospheric planet of interest. Together with the TPS mass obtained from TPSSIZER and the propellant mass estimate from the trajectory tool POST2, the weight/sizing closure analysis code, XWAT/XClosure [9] provides an estimate of the delivered payload for the particular vehicle shape parameters selected by the MDO algorithm. Optimization of this delivered payload is then the objective function being maximized by the COBRA MDO algorithm.

An overview of the COBRA process described above is shown in Figure 2. The MDO driver will then find a Pareto front amongst the allowed range of all of the vehicle shape parameters being considered, which then can be further explored. Typically, a global optimization process with sparse population using engineering fidelity analysis at a single trajectory point is used to explore the initial vehicle design space. This is followed by a full vehicle design including trajectory and TPS sizing optimization and finally weights and sizing to perform the final vehicle closure. A less densely populated design space is generated next using high-fidelity analyses anchored to engineering analyses. It is at this point in the optimization process where the use of parallel computing becomes extremely necessary to further explore the design space in a reasonable turnaround time.

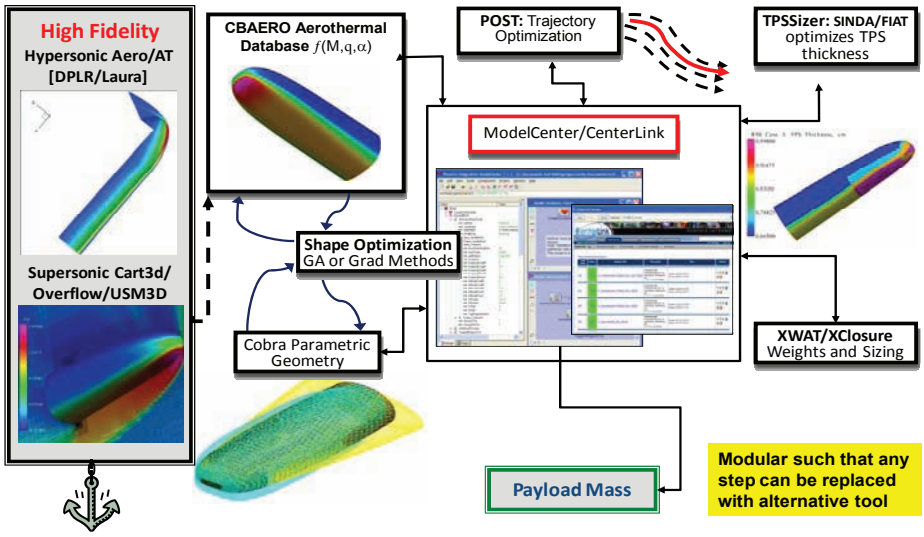


Figure 2: Co-Optimization Bluntbody Re-entry Analysis Process

3. RESULTS

In order to test and validate the integrated MDO process, the Mars Pathfinder (MPF) entry probe vehicle was used. MPF is chosen since it was instrumented to transmit trajectory and thermal couple (TC) data during its entry [10]. The Pathfinder vehicle was an axi-symmetric 70.2 deg. half-angle blunted cone with a rounded shoulder and a truncated 46.6 deg. conical back shell. Based on the Pathfinder vehicle geometry specifications, a pre-existing surface mesh is uploaded into the COBRA framework using an option for user provided mesh. With this, the COBRA framework is run to provide the aero-thermodynamic database (ADB) from the CBAERO engineering tool execution which provides both the aerodynamics for the trajectory analysis performed by the POST2 trajectory analysis tool, and the aerothermodynamics data for the thermal protection system sizing, performed by the TPSSizing tool. Results of this process are shown in Figures 3-4.

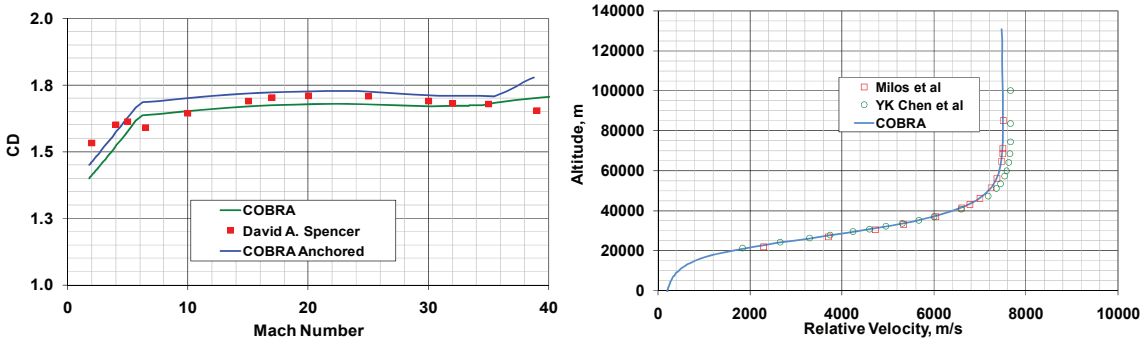


Figure 3: Mars Pathfinder (a) drag coefficient profile and (b) trajectory comparison

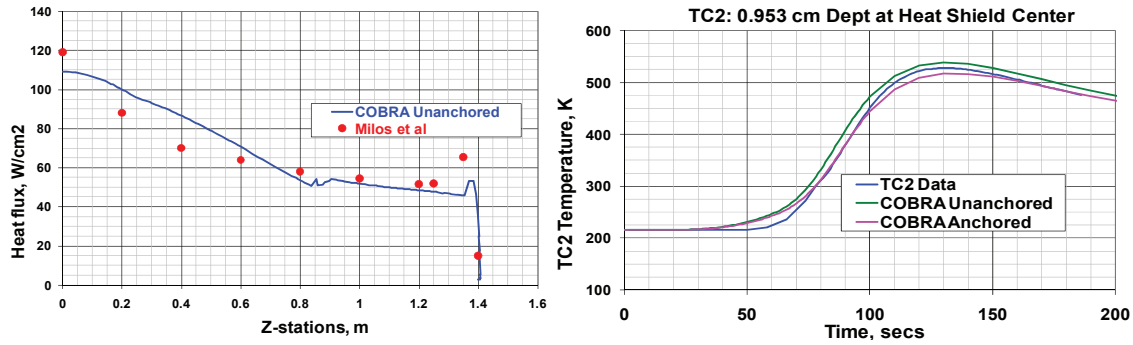


Figure 4: Mars Pathfinder centreline peak heating and thermal couple comparison

Figure 3 shows good agreement between the predicted drag coefficients and trajectories as compared to previously published data [10-12]. Figure 4a shows the Pathfinder peak heating along the centerline of the vehicle and good agreement with high fidelity CFD from previously published data [10]. Finally, Fig.4b shows a comparison of the thermal couple temperature data near the nose of the vehicle and good agreement with the flight data is seen. These comparisons provide confidence that the individual disciplinary analysis tools integrated within the MDO framework predict reasonable results for the design of atmospheric entry vehicles.

Next, the results of utilizing the COBRA MDO process for a hypothetical mission to land the maximum payload possible on the surface of Mars utilizing the Delta IV launch vehicle, is described. Based on an excess launch energy, C3, value of 10 km^2/s^2 which is an appropriate value for achieving a Mars mission, the estimated transfer mass from Earth to Mars entry interface is estimated to be approximately 8800 kg. This number was taken from the Delta IV planner's guide [13]. Accounting for the vehicle adapter mass (~425kg) and cruise stage mass (~630kg) a useful arrival mass of 7745kg at Mars entry interface is estimated. To be conservative, an entry vehicle total mass of 7,500 kg was selected. The above mission parameters define our MDO global objective of maximizing the payload mass placed on the planet surface of Mars and provide us with a constraint on what the outer mode line shape of our entry vehicle can be based on the launch vehicle shroud for the Delta IV vehicle. The parametric shape code has pre-coded the various launch vehicle shrouds so as to provide an interference check to prevent violating this constraint.

In order to explore the shape parameter space within the MDO framework, the ModelCenter Genetic Algorithm optimizer, Darwin, is used. The constraints for this optimization are "number of triangles which violate the launch faring"=0, and "Cmα"<0. The launch fairing triangle constraints ensures the shapes do not violate the launch fairing. The Cmα constraint is an aerodynamic constraint to ensure that the vehicles being explored are statically stable in the pitch plane. The objectives are to minimize the convective heating on the vehicle (qdot_convection) and maximize the drag area (CDa) which correlates directly to how quickly the vehicle will slow down before reaching the supersonic parachute deployment constraint [14] where higher CDa is desirable. The results of this optimization are plotted in Fig. 5a. Note that the "✚" symbols denote the "Pareto front." Any point on the plot in Fig. 5a is a potential solution and the Pareto front are the solutions which best meet the multiple objectives. For our purposes the point denoted in Fig. 5 with the red arrow was chosen as the vehicle shape to explore further. This shape and its hypersonic pressure distribution are shown in Fig. 5b.

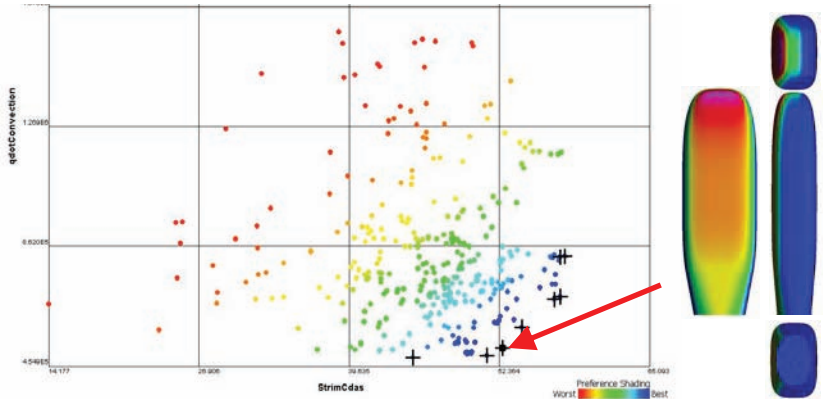


Figure 5 (a) GA shape optimization results qdot convection vs. CdA and (b) Shape chosen.

After seeing the optimized shape, it was realized that the aft of the vehicle was tapered and may not be directionally stable for the center of gravity (C.G.) location selected. The C.G. location is selected by restricting its placement along the centerline (y=0, z=0) of the vehicle and its x-axis placement is determined so as to trim the vehicle to fly at an angle-of-attack that achieves a desired lift-to-drag (L/D) ratio. A quick analysis was done which verified that the optimized shape was not directionally stable for the C.G. location chosen. Due to the flexibility of the shape code, the aft portion of the vehicle shape was modified to avoid the taper and still maintain the overall nose and lower surface shape. The result was a directionally stable vehicle with very similar aero/aerothermal performance. It should be noted that the optimization did not have a constraint to check for directional stability. This was added in and is now a constraint for future optimizations.

Next, the aeroheating was looked at. It was decided that the small rounded corners on the nose may be problematic and a variation on the parameters which control the corner radius, PowN and PowX, were varied to explore nose shapes that showed lower heating without penalizing the aerodynamic performance. The outcome of this is what we

call engineering alternative five (EA5) which showed lower total heating (Qdot) with similar shape parameters and aero performance to the original optimized shape. Finally, the EA5 shape was used to complete the MDO design analysis by generating a full aerothermodynamic database with CBAERO that is used as input into the POST trajectory optimization tool. The POST trajectory for this vehicle is optimized to minimize heat load while meeting the supersonic parachute deployment window constraint and the g-load constraints. With this trajectory the thermal protection system is optimized with the TPSSizer tool and the results are shown in the table below.

Concepts	Area [m^2]	*TPS_Wt. [kg]	TPS_UWT [kg/m2]
COBRA-EA5	175.2	451.6	2.58

In the final paper we will further demonstrate this capability. Also, in the final paper, the weights and sizing analysis tools will be integrated to provide a prediction for the final payload mass capability.

4. CONCLUSIONS

A Multi-disciplinary Design Optimization process has been developed for planetary atmospheric entry vehicle concepts accounting for shape, trajectory, thermal protection system (TPS) and vehicle closure in order to simultaneously satisfy desirable hypersonic aerothermodynamic, supersonic aerodynamic, and TPS characteristics. This has been accomplished using a combination of engineering and higher-fidelity physics analysis tools along with optimization methods and engineering judgment. Due to the increased computation intensive aspect of using higher fidelity physics based codes, the parallelization of individual tools and the entire MDO process has proven to be necessary in providing efficient turnaround of multiple entry vehicle designs.

ACKNOWLEDMENTS

ELORET staff support from Xun Jiang and Eric Lau has been crucial for the integration of the COBRA framework and greatly appreciated. Inputs from Nagi Mansour of NASA Ames Research Center are greatly appreciated.

REFERENCES

1. Brown, J. L., Garcia J. A., Kinney, D. J., and Prabhu, D. K., "An Asymmetric Capsule Vehicle Geometry Study for CEV" *45th AIAA Aerospace Sciences Meeting and Exhibit* 2007, Reno, Nevada
2. Kinney D., "Aero-Thermodynamics for Conceptual Design," AIAA-2004-31*, 42nd AIAA Aerospace Sciences Meeting and Exhibit*, Reno NV, Jan. 2004.
3. Wright, M J, Candler, G, and Bose, D, "Data-Parallel Line Relaxation Method for the Navier-Stokes Equations," *AIAA J.,* Vol. 36, No. 9, 1998, pp. 1603-1609.
4. Kinney D., "Aerothermal Anchoring of CBAERO Using High Fidelity CFD," AIAA-2007-0608, *45th AIAA Aerospace Sciences Meeting and Exhibit*, Reno NV, Jan. 2007.
5. G.L. Brauer,et al, "Capabilities and Applications of the program to Optimize Simulated Trajectories (POST)," NASA CR-2770, February 1977.
6. anon.,The NASTRAN Programmer's Manual, December 1978 Edition, *Scientific and Technical Information Office, National Aeronautics and Space Administration*, 1978.
7. McGuire M. K., Bowles J., Yang L., Kinney D., Roberts C., "TPS Selection & Sizing Tool Implemented in an Advanced Engineering Environment," AIAA-2004-342, *42nd AIAA Aerospace Sciences Meeting and Exhibit*
8. Chen, Y.-K., and Milos, F.S., "Ablation and Thermal Analysis Program for Spacecraft Heatshield Analysis," *J. Spacecraft and Rockets*, vol. 36, No. 3, 1999, pp. 475-483.
9. X. Jiang, P. Gage, J.C. Vander Kam, M. Qu, "Weights Analysis of Space Launch Vehicles in an Advanced Engineering Environment", *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference 2004*,
10. Frank S Milos and Y-K. Chen, "Mars Pathfinder Entry Temperature Data, Aerothermal Heating, and Heatshield Material Response," *Journal of Spacecraft and Rocket*, Vol.36, No.3, 1999.
11. David A. Spencer and Robert D. Braun, "Mars Pathfinder Atmospheric Entry Design," AAS 95-379
12. Chen, Y.K., Henline, W.D., and Tauber, M.E., "Mars Pathfinder Trajectory-Based Heating and Ablation Calculations," *Journal of Spacecraft and Rocket*, Vol.32, No.2, 1995, pp.225-230
13. Delta IV Payload Planners Guide, United Launch Alliance, September 2007
14. Braun, R. D. and Manning, R. M., "Mars Exploration Entry, Descent and Landing Challenges," Paper 1076, *2006 IEEE Aerospace Conference*, Big Sky, Montana, March 2006.

OTHER APPLICATIONS

Numerical modeling of nonequilibrium driven cavity gas flow with a high-order moment approach

Xiao-Jun Gu[1], David R. Emerson[2], Gui-Hua Tang[3], Charles Moulinec[4]

Computational Science and Engineering Department,
STFC Daresbury Laboratory, Warrington, WA4 4AD, UK
[1](*email: xiaojun.gu@stfc.ac.uk*)
[2](*email: david.emerson@stfc.ac.uk*)
[3](*email: guihua.tang@stfc.ac.uk*)
[4](*email: charles.moulinec@stfc.ac.uk*)

Abstract: A high-order moment approach is employed to model gas flow in a driven cavity under nonequilibrium conditions. To apply the moment method to low speed flow, the higher moments are decomposed into components representing their gradient (hydrodynamic) and nongradient (nonhydrodynamic) parts. As a result, the moment equations can be expressed in a convection-diffusion form and traditional CFD techniques for low speed flow can be used. The moment equations have been implemented in our in-house parallel CFD code, THOR. The validity of the moment method for nonequilibrium gas flow in the transition regime is assessed and the parallel performance of the code is measured on different computer platforms.

Keywords: nonequilibrium flow, driven cavity, moment method, Knudsen number

1. INTRODUCTION

Gases in a low density environment or a microsystem will exhibit nonequilibrium phenomena because the number of molecular collisions is not sufficiently large for the gas to reach an equilibrium state. The extent of the nonequilibrium can be measured by the Knudsen number, Kn, which is the ratio of the molecular mean free path to the characteristic length of interest. A nonequilibrium gas can readily be described by kinetic theory and the Boltzmann equation [1] or simulated by the direct simulation Monte Carlo (DSMC) method [2]. However, the numerical solution of the Boltzmann equation for practical applications remains formidable due to the complicated structure of the collision term and its high dimensionality. It is also computationally expensive to use DSMC in the regime not far away from the equilibrium state, i.e. $Kn < 1$, particularly for low speed flows. On the other hand, the conventional macroscopic description, given by the Navier-Stokes-Fourier (NSF) equations, is no longer able to provide an accurate description of gas flow in the early transition regime (i.e. $0.1 \le Kn \le 1.0$). Alternative macroscopic modeling and simulation strategies [3-6] have been developed for many years but in this work, the Grad moment method [7], which was developed originally as an approximate solution procedure to the Boltzmann equation, is adopted to model driven cavity flow in the early transition regime.

In Grad's method of moments, a set of moments of the molecular distribution function is defined to describe the state of the gas. For a local equilibrium gas, the state of the gas can be determined by the first five lowest moments, density ρ, temperature T, and velocity u_i. As the gas departs from the local equilibrium state, additional moments are required. Grad was one of the pioneers to introduce the stress, σ_{ij}, and heat flux, q_i, as extended macroscopic variables and derived the governing equations for them from the Boltzmann equation. This results in the well-known Grad 13 moment equations. Recently, Grad's 13 moment equations have been regularized [8, 9] to overcome some of their limitations, such as sub-shock structure [10]. With a set of newly derived wall boundary conditions [11, 12], the regularized 13 moment equations (R13) have been applied to confined microsystem problems. The Knudsen layer is an important feature in micro-scale gas flow. In the moment method, the Knudsen layer appears as superpositions of exponential layers [13]. The R13 equations are the smallest moment system which describes the gradient and nongradient transport modes in the transition regime and provide only one such layer contribution. To improve the accuracy of the moment method, a regularized 26 moment system [14, 15] has been derived by the authors. The R26 equations accurately capture the nonequilibrium phenomena predicted by kinetic theory, such as the Knudsen layer velocity, Knudsen minimum [16], nongradient heat flux, and bimodal temperature profile [17, 18] in the one dimensional planar Couette and Poiseuille flow for Knudsen numbers up to unity.

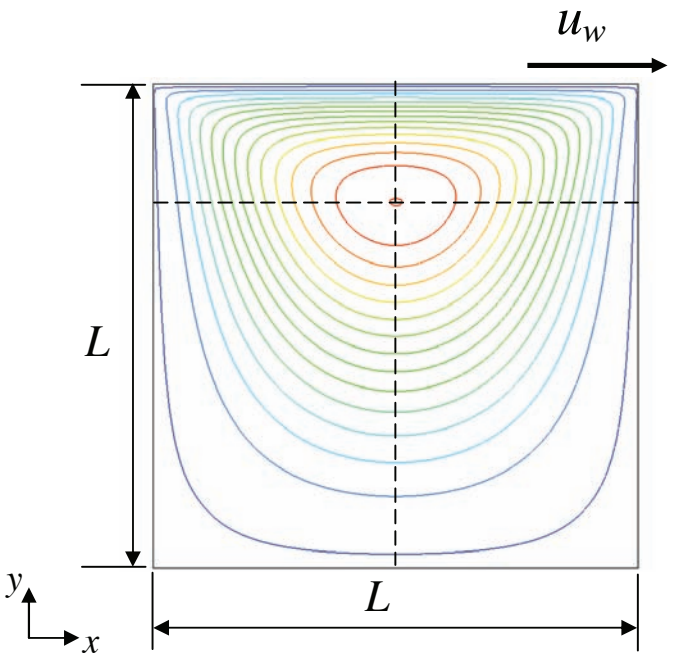


Fig 1 Configuration of driven cavity flow

2. EXTENDED HYDRODYNAMIC MODEL- THE REGULARIZED 26 MOMENT EQUATIONS

Combining Grad's moment method [7] and the regularization procedure [8], a system of 26 moment equations for monatomic gas flow has been developed by the authors [14, 15]. In addition to conservation laws and the governing equations for the stress, σ_{ij}, and heat flux term, q_i, the governing equations for the higher moments m_{ijk}, R_{ij} and Δ are included in the 26 moment system to account for nonequilibrium effects. The set of moment equations will be listed in the full paper and a detailed derivation of these equations can be found in [15].

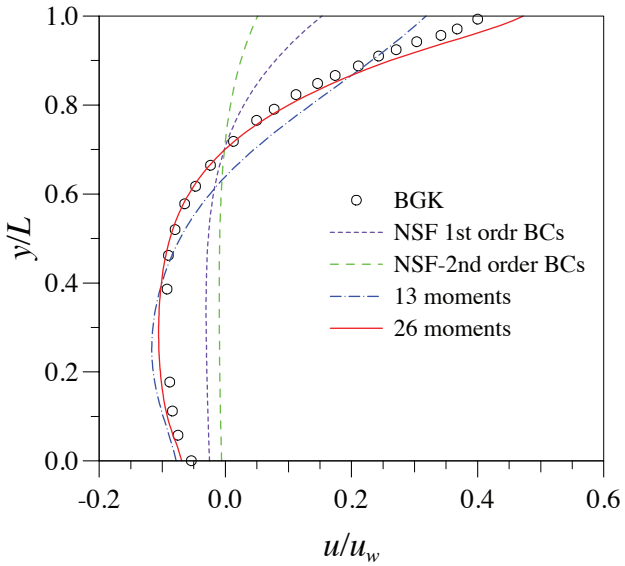


Fig. 2 Predicted horizontal velocity u in the vertical plane through the vortex centre

3. NUMERICAL METHOD AND PARALLELISATION

The original Grad's moment equations are a set of first-order partial differential equations with hyperbolic characteristics and the regularized moment equations are a mixed system of first and second order partial differential equations. Traditionally, these equations are used to study hyperbolic flows [10, 19]. In the case of low speed gas flow, the flow is parabolic or elliptic. Using a hyperbolic flow solver to solve elliptic flows is inefficient and expensive. To overcome this issue, the moments are decomposed into their gradient (hydrodynamic) and nongradient (non-hydrodynamic) transport parts [11, 14, 15] defined by:

$$\sigma_{ij} = \sigma_{ij}^G + \rho g_{ij},\ \ q_i = q_i^G + \rho h_i,\ \ m_{ijk} = m_{ijk}^G + \rho\omega_{ijk},\ \ R_{ij} = R_{ij}^G + \rho\gamma_{ij} \ \text{ and } \ \Delta = \Delta^G + \rho\chi \tag{1}$$

where ρg_{ij}, ρh_i, $\rho\omega_{ijk}$, $\rho\gamma_{ij}$ and $\rho\chi$ are the nongradient components for σ_{ij}, q_i, m_{ijk}, R_{ij} and Δ, respectively and σ_{ij}^G, q_i^G, m_{ijk}^G, R_{ij}^G and Δ^G are the gradient components. This decomposition, which was originally proposed to study the R13 equations [11], is nonlocal and symmetric. As a result, the moment equations can be expressed in convection-diffusion form.

$$\underbrace{\frac{\partial\rho\Phi}{\partial t}}_{\text{transition}} + \underbrace{\frac{\partial\rho u_l\Phi}{\partial x_l}}_{\text{convection}} - \underbrace{\frac{\partial}{\partial x_l}\left(\frac{\mu}{\Gamma_\Phi}\frac{\partial\Phi}{\partial x_l}\right)}_{\text{diffusion}} = \underbrace{S_\Phi}_{\text{source}} \tag{2}$$

in which $\Phi = (u_i, T, g_{ij}, h_i, \omega_{ijk}, \gamma_{ij}, \chi)$, $\Gamma_\Phi = (1, 2/5, 3/2, 5/6, C_1, 7Y_1/9, 3/7)$ and $S_\Phi = (S_{u_i}, S_T, S_{g_{ij}}, S_{\gamma_{ij}}, S_{h_i}, S_{\omega_{ijk}}, S_\chi)$ corresponds to the source terms of the respective governing equations. These equations form a set of second order partial differential equation. The mathematical characteristics of the system will be determined by the flow conditions. They are of a hyperbolic nature for high speed flows and parabolic or elliptic when the flow velocity is low or the flow is re-circulating. In this way, traditional CFD techniques for low speed flows can be used. In the present study, the finite volume approach has been employed. The diffusion and source terms are discretized by a central difference scheme. For the convective terms, a range of upwind schemes are available. The SIMPLE algorithm is adopted to resolve the coupling of the velocity and pressure fields. A collocated grid arrangement is used and any non-physical pressure oscillations are eliminated by the interpolation scheme of Rhie and Chow. The 26 moment equations are implemented in the in-house code THOR, which is fully parallelized with MPI and grid partitioning methods.

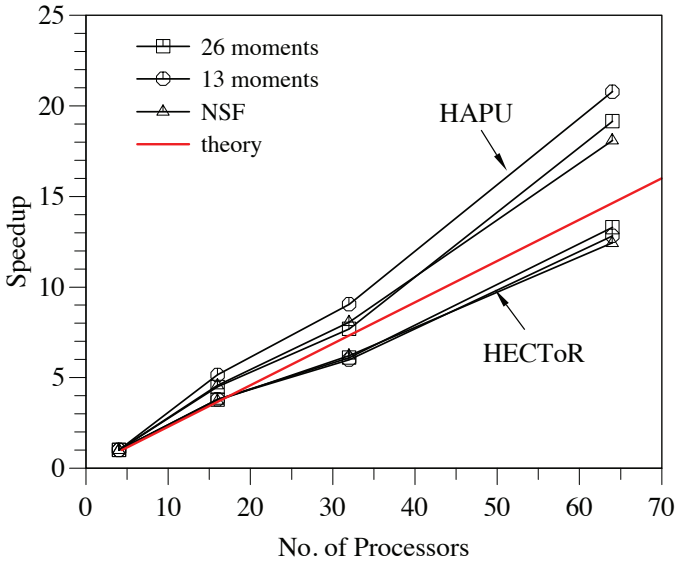


Fig. 3 The speedup of THOR on HAPU and HECToR

4. RESULTS AND DISCUSSIONS

In the present study, driven cavity flow in the transition regime is computed with the different moment systems to assess their validity in a 2D geometry. The configuration of the square driven cavity is shown in Fig. 1. The size of the square cavity is L and the lid velocity is u_w. The rest of the walls are stationary. The Knudsen number is defined by $Kn = (\mu / pL)\sqrt{\pi RT/2}$, where μ is the viscosity, p is the pressure and T is the temperature. To be consistent with the conditions studied by kinetic theory [20], the Reynolds number and Mach number is very small. The computed velocity profiles from the NSF equations with first and second order slip-boundary conditions, the R13 and R26 moment equations are presented in Fig. 2 in comparison with the results from the BGK kinetic equation [20] at $Kn = 0.886$. In general, the NSF equations with both first and second order slip-boundary conditions are unable to give qualitatively correct velocity profiles in both directions. As expected, the R26 equations predict the velocity profiles more accurately than the R13.

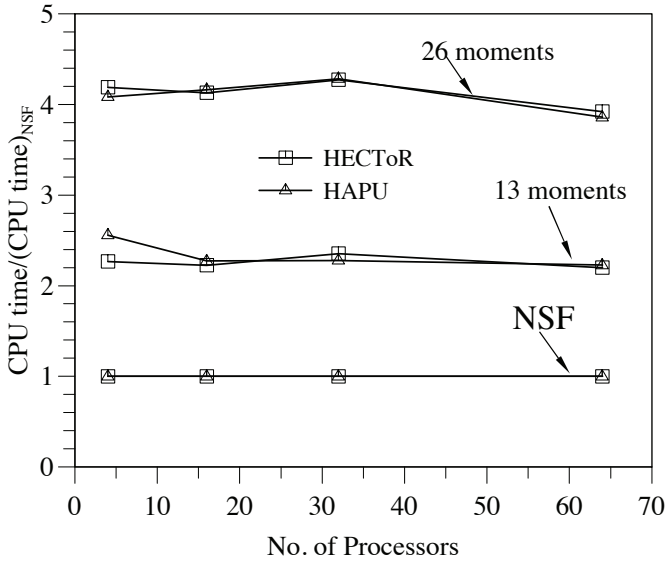


Fig. 4 CPU time for higher order moment equations normalised by CPU time for NSF equations.

The performance of the code is measured on two different parallel platforms, HAPU and HECToR. HAPU is a local HP Cluster Platform 4000 based on Redhat Enterprise Linux 5. It has 128 x 2.4GHz AMD Opteron cores, with 2Gb memory per core and a Voltaire InfiniBand interconnect. HECToR is the UK's front-line national supercomputing service. The HECToR Phase 1 configuration is an integrated system known as “Rainier”, which includes a scalar MPP XT4 system, a vector system known as “BlackWidow”, and storage systems. The Cray XT4 scalar supercomputer comprises 1416 compute blades, each of which has 4 dual-core processor sockets. This amounts to a total of 11,328 cores, each of which acts as a single CPU. The processor is an AMD 2.8 GHz Opteron. Each dual-core socket shares 6 GB of memory, giving a total of 33.2 TB in all. Each dual-core socket controls a Cray SeaStar2 chip router. This has 6 links which are used to implement a 3D-torus of processors. The point-to-point bandwidth is 2.17 GB/s, and the minimum bi-section bandwidth is 4.1 TB/s. The latency between two nodes is around 6μs.

A grid size of 1024x1024 is used to study the code’s parallel performance. The speedup of the code running different models is shown in Fig. 3. Although THOR is scaling linearly for all three models on both HAPU and HECToR up to 64 processors, the speedup of the code on HECToR is only two-thirds of that on HAPU. Possible reasons for the differences observed with HECToR are the faster CPUs and any dual-core network contention. Another reason for the superlinear behaviour on HAPU could be due to cache effects, which do not occur on HECToR.

Shown in Fig. 4 is the normalized CPU time for the higher moment systems. To solve the 13 moment equations, it requires more than twice the CPU time of the NSF equations. The CPU time required for solving the 26 moment equations is doubled again over the 13 moment equations and is more than four times the CPU time for the NSF equations. As the number of processors increases, these ratios do not change.

ACKNOWLEDGEMENTS

The authors would like to thank the Engineering and Physical Sciences Research Council (EPSRC) for their support of Collaborative Computational Project 12 (CCP12).

REFERENCES

1. Cercignani, C. (1988). *The Boltzmann Equation and Its Applications,* Springer, New York.
2. Bird, G. (1994). *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, Claredon Press, Oxford.
3. Chapman, S. and Cowling, T. G. (1970). *The Mathematical Theory of Non-uniform Gases*, Cambridge University Press.
4. Struchtrup, H. (2005). *Macroscopic Transport Equations for Rarefied Gas Flows*, Springer-Verlag, Berlin-Heidelberg.
5. Muller, I. and Ruggeri, T. (1993). *Extended Thermodynamics*, Springer-Verlag, New York.
6. Xu, K. (2001). A gas-kinetic BGK scheme for the Navier-Stokes equations and its connection with artificial dissipation and Godunov method. *J. Comput. Phys.* 171:289-335.
7. Grad, H. (1949). On the Kinetic Theory of Rarefied Gases, *Commun. Pure Appl. Math.* 2:331-407.
8. Struchtrup, H. and Torrihon, M. (2003). Regularization of Grad's 13 moment equations: Derivation and linear analysis. *Phys. Fluids* 15:2668-2680.
9. Struchtrup, H. (2004). Stable transport equations for rarefied gases at higher orders in the Knudsen number. *Phys. Fluids* 16:3921-3934.
10. Grad, H. (1963). Asymptotic Theory of the Boltzmann Equation, *Phys. Fluids* 6:147-181, 1963.
11. Gu, X. J. and Emerson, D. R. (2007). A computational strategy for the regularized 13 moment equations with enhanced wall-boundary conditions. *J. Comp. Phys.* 225:263-283.
12. Torrilhon, M. and Struchtrup, H. (2008). Boundary conditions for regularized 13-moment-equations for micro-channel-flows, *J. Comput.Phys.* 227:1982-2011.
13. Struchtrup, H. (2008). Linear kinetic transfer: moment equations, boundary conditions, and Knudsen layer. *Physica A* 387:1750-1766.
14. Gu, X. J. and Emerson, D. R. New approaches to modeling rarefied gas flow in the slip and transition regime. Invited Lecture, International Conference on Parallel Computational Fluid Dynamics, May 21-24 2007, Antalya, Turkey.
15. Gu, X. J. and Emerson, D. R. (2009). A high-order moment approach for capturing non-equilibrium phenomena in the transition regime, accepted for publication in *J. Fluid Mech.*.
16. Ohwada, T., Sone, Y. and Aoki, K. (1989). Numerical analysis of the Poiseuille and thermal transpiration flows between two parallel plates on the basis of the Boltzmann equation for hard-sphere molecules. *Phys. Fluids A* 1:2042-2049.
17. Uribe, F. J. and Garcia, A. L. (1999). Burnett description for plane Poiseuille flow. *Phys. Rev. E* 60:4063-4078.
18. Zheng, Y., Garcia, A. L. and Alder, B. J. (2002). Comparison of kinetic theory and hydrodynamics for Poiseuille flow. *J. Stat. Phys.* 109:495-505.
19. Torrilhon, M. and Struchtrup, H. (2004). Regularized 13 moment equations: Shock structure calculations and comparison to Burnett models, *J. Fluid Mech.* 513:171-198.
20. Naris, S. and Valougeorgisa, D. (2005). The driven cavity flow over the whole range of the Knudsen number. *Phys. Fluids* 17, 097106.

Simulation of the climate of the XX century in the Alpine space: numerical tests on NEC SX-9 supercomputer

E. Bucchignani*, R. Mella, P. Mercogliano***, P. Schiano******

*CIRA Centro Italiano Ricerche Aerospaziali, 81043 Capua CE, Italy
(Tel: +39 0823 623725; e-mail: e.bucchignani@cira.it)
** CIRA Centro Italiano Ricerche Aerospaziali, 81043 Capua CE, Italy
(e-mail: r.mella@cira.it)
*** CIRA Centro Italiano Ricerche Aerospaziali, 81043 Capua CE, Italy
(e-mail: p.mercogliano@cira.it)

**** CIRA Centro Italiano Ricerche Aerospaziali, 81043 Capua CE, Italy
(e-mail: p.schiano@cira.it)

Abstract: This paper deals with the numerical simulation of the climate of the XX century by means of the regional climate model COSMO-CLM. The area considered is the Alpine space that, with its complex orography, needs to be described with a non-hydrostatic model at high resolution. Numerical tests are performed on the NEC SX-9 supercomputer; particular attention is given to the study of the precipitations.

Keywords: Regional Climate Modeling, Atmospheric processes

1. INTRODUCTION

The usage of a Regional Climate Model (RCM) with an horizontal resolution of around 10 *km* for an area like the Alpine Space, with its complex orography, can be a useful tool for the description of the precipitation patterns expected in the next century especially in small-size (10-100 km^2) Alpine river catchments, which are characterized by short response times, sometimes less than 6 hours, to intense extreme precipitation events (the basin response time is the time necessary for the water to flow through the drainage system); this means that there is very brief time between the precipitation event and the possible flood that follows. The knowledge of very precise expected trends for the precipitation patterns can help the decision makers to prepare mitigation and adaptation strategies; in fact intense rains are the causes of floods, but also of some particular landslides typology (as mudflow). The high attention of the scientific and social community for these applications is due to the expected increasing of natural hazards for effect of climate changes. In order to assess the climatic changes on this area we therefore need information at the smallest possible spatial scales. The global climate model used in this application, ECHAM4 [1], is characterized by a resolution of about 100 *km*, which is too coarse for impact studies on this area, where these phenomena have a characteristic space scale of hundreds of meters. The regional climate model COSMO-CLM [2] can be used for simulations on time range up to a century, with a spatial resolution between 1 and 50 *km*; at such resolutions, terrain height is better described (Figure 1) with respect to the global model, where there is in over-/underestimation of valley/mountain heights resulting in errors for precipitation estimation, which is closely related to terrain height. It is well known that climate codes require high computational resources in terms of memory and CPU time and therefore the use of a multiprocessor machine is mandatory. In this work, the regional climatic system COSMO-CLM has been employed: it is a non-hydrostatic model for the simulation of atmospheric processes. It has been developed by the DWD–Germany and by the COSMO consortium for weather forecast services [3]. Successively, the model has been updated by the CLM-Community, in order to develop also climatic applications.

The non-hydrostatic modelling allows providing a good description of the convective phenomena, which are generated by vertical movement (through transport and turbulent mixing) of the properties of the fluid as energy (heat), water vapour and momentum.

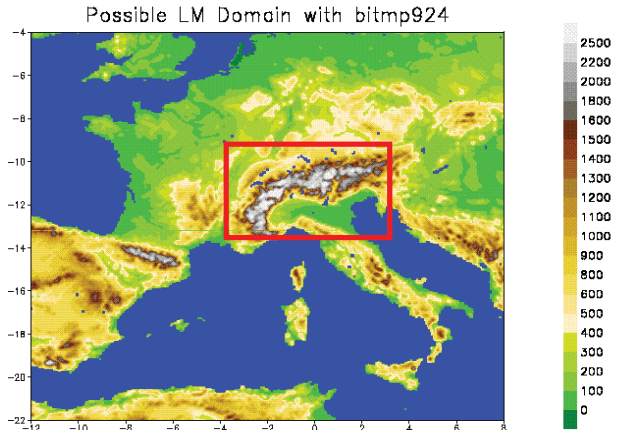


Fig. 1: CLM orography using a horizontal resolution of 7 km. In the red box the Alpine space is contained.

Convection can redistribute significant amounts of moisture, heat and mass on small temporal and spatial scales. Furthermore convection can cause severe precipitation events (as thunderstorm or cluster of thunderstorm). Being a subgrid scale phenomenon, the convection usually has to be parameterised. In the CLM code, different convection scheme are implemented as Tiedtke scheme [4], which is also used in the reference version, and the Kain-Fritsch scheme [3]. The choice of this parameterization is essential in order to simulate the convective phenomena as the strong convective summer precipitation; these last ones, in fact, may cause flash floods because the heavy rain falling on bare soil and rocks runs off much faster than in winter, when the vegetation is present. Results obtained using these 2 different scheme will be shown. The mathematical formulation of COSMO-CLM is made up of the Navier-Stokes equations for a compressible flow. The atmosphere is treated as a multicomponent fluid (made up of dry air, water vapour, liquid and solid water) for which the perfect gas equation holds, and subject to the gravity and to the Coriolis forces. The model includes several other "parameterizations", in order to keep into account, at least in a statistical manner, several phenomena that take place on unresolved scales, but that have significant effects on the meteorological interest scales (for example, interaction with the orography). Further parameterizations are available in order to describe some important physical phenomena for the atmospheric evolution, for example solar radiation, precipitations, soil behaviour and microphysics.

The governing equations can be written as follows [5]:

$$\rho \frac{d\vec{v}}{dt} = -\nabla p + \rho g - 2\Omega \times (\rho \vec{v}) - \nabla \cdot \bar{t}$$

$$\frac{dp}{dt} = -(c_p / c_v) p \nabla \cdot v + (c_p / c_v - 1) Q_h + (c_p / c_v) Q_m$$

$$\rho c_p \frac{dT}{dt} = \frac{dp}{dt} + Q_h$$

$$\rho \frac{dq^x}{dt} = -\nabla \cdot J^x + I^x$$

$$p = \rho R_d (1 + \alpha) T$$

In which $\vec{v}$ is the velocity vector, ρ is the density, p is the pressure, g is the gravity acceleration, Ω is the constant angular velocity of earth rotation, $\vec{t}$ is the stress tensor due to viscosity, c_v and c_p are the specific heat of moist air, respectively at constant volume and constant pressure. Q_h and Q_m are respectively the heat production per unit volume of air and the impact of changes in the concentrations of the humidity constituents on the pressure tendency.

Moreover, T is the temperature, q^x are the mass fractions of the constituents of the mixture (i.e. $x = d, v, l, f$ respectively for dry air, water vapour, liquid water and frozen water), J_x and I_x are respectively the source/sink and the diffusion flux of the constituent x and R_d is the gas constant for dry air. The moisture term α is defined as $\alpha = (R_v / R_d - 1)q^v - q^l - q^f$, where R_v *i*s the gas constant for water vapour.

COSMO-CLM considers limited domains, so it is not possible to impose physical boundary conditions, therefore they are obtained in a numerical manner by means of a dynamical *downscaling* technique, from a global climatic model. In this work, boundary conditions are obtained from the results of ECHAM4 global model [1]. This circumstance limits the adoptable spatial resolution, because the resolution ratio (global/regional) cannot be larger than eight. Initial conditions can be calculated on the basis of experimental observations made by survey stations.

The discretization of the fluid dynamics equations is performed by using finite difference approximation, on a computational grid defined in a rotated spherical coordinate system. The pole is tilted and can be positioned such that the equator runs through the centre of the model domain. Problems resulting from the convergence of the meridians can be minimized for any limited area model domain on the globe. Especially, for a very small domain with negligible impact of the curvature of the earth's surface, the equations become identical to those for a tangential Cartesian coordinate system. Three time integration algorithms are available: the first one is based on a second order accurate Runge-Kutta method on two time levels; the second is based on the “horizontal explicit - vertical implicit” variant of Leapfrog scheme, the third based on a semi-implicit Leapfrog scheme on three time levels. The parallelization is done by horizontal domain decomposition using a soft-coded 2-gridline halo. The Interface software MPI is used as Message Passing.

The numerical simulations will be performed on the new NEC SX-9 supercomputer, installed at CMCC-Lecce (Italy). It is a vector/parallel multiprocessors machine with 7 nodes, each of which having 16 processors, for a global number of 112 processors. Each processor has a peak performance of 102.4 GFlops, so the peak performance is 11.4 TFlops; the system has 3,5 TBytes of central memory. The intranode data transfer rate is 4 TBytes/sec, while the internode data transfer rate is 2x128 GBytes/sec. The system has 50 TBytes of disk space. The operating system is NEC SuperUX 18.1, while *Fortran90, C/C++* compilers are available. This machine is ideal for weather forecasting, fluid dynamics and environmental simulation, as well as simulations for as-yet-unknown materials in nanotechnology and polymeric design. Performances of COSMO-CLM, in terms of parallel speed-up, will be given. Different numerical experiments performed with CCLM in the Alpine Space will be presented in order to evaluate the model performances on this area; particular attention will be given to the study of the precipitations. Some simulations will be performed also on a cluster of 30 IBM P575 nodes, installed at CMCC-Lecce. Each node has 32 Power6 (4.7GHz) cores, for a global number of 960 cores. The peak performance is about 18 TFlops; each node has 128 GB of memory. The cluster is characterized by Infiniband 4x DDR interconnection, while the operating system is AIX v.5.3.

REFERENCES

1. Gualdi S., Scoccimarro E., Navarra A. (2008). Changes in Tropical Cyclone Activity due to Global Warming: Results from a High-Resolution Coupled General Circulation Model, *Journal of Climate* **21**, 5204-5228.
2. Steppeler J., Doms G., Schättler U., Bitzer H.W., Gassmann A., Damrath U., Gregoric G. (2003). Meso-gamma scale forecasts using the nonhydrostatic model LM, *Meteorol. Atmos. Phys*., **82** 75-96.
3. Kain J.S., Fritsch J.M. (1993). Convective parameterization for mesoscale models: the Kain-Fritsch scheme, *Meteorol. Monographs* **24**, 165-170.
4. Tiedtke M. (1989). A comprehensive mass flux scheme for cumulus parameterization in large scale models. *Mon. Weather Rev.* **117**, 1779-1800.
5. Holton J.R. (2004). *An introduction to Dynamic Meteorology,* Elsevier Academic Press.

Development of a Virtual Mesh Refinement Algorithm in a Parallel Unstructured-grid DSMC Code

C.-C. Su*, K.-C. Tseng, J.-S. Wu*, J.-P. Yu***, Y.-Y. Lian****

*Department of Mechanical Engineering, National Chiao Tung University, Hsinchu, TAIWAN
(Tel: 886-3-573-1693; e-mail: chongsin@facu;ty.nctu.edu.tw)
**National Space Organization, Hsinchu, TAIWAN
(e-mail: kctseng@nspo.org.tw)
*** Department of Information Management, Ming Chuan University, Taipei, TAIWAN
(e-mail: jpyu@mail.mcu.edu.tw)

Abstract: In this paper, a virtual mesh refinement (VMR) algorithm for unstructured grids is presented in a parallelized direct simulation Monte Carlo code (PDSC) which features transient adaptive sub-cell (TAS). This algorithm is a virtual mesh refining process, in which the background mesh is refined based on an initial DSMC simulation. The refined cells are designed in a way similar to the structured grid, which makes the particle tracing on them very efficient, unlike on unstructured grids. These refined cells are only used for particle collision and sampling to physically resolve the collision mechanics. Only a refined cell, which includes centroid of the background cell, in a background cell is used for outputing macroscopic data. Two hypersonic flows including a hypersonic cylinder flow and a scramjet flow are simulated. Results show that the simulations using VMR can faithfully reproduce the benchmark case with a much reduced computational time. Corresponding parallel performance of this VMR in PDSC up to 128 processors will be presented in the conference.

Keywords: virtual mesh refinement, parallel direct simulation Monte Carlo, unstructured grid.

1. INTRODUCTION

The direct simulation Monte Carlo (DSMC) method is a computational tool for simulating flows in which effects at the molecular scale become significant [1]. The computational domain itself is divided into either a structured or unstructured grid of cells which are then used to select particles for collisions on a probabilistic basis and also are used for sampling the macroscopic flow properties. The method has been shown to provide a solution to the Boltzmann equation as the number of simulated particles tends toward the true value within the flow field [2]. The DSMC cells are used for particle collisions and sampling of macroscopic properties, in which the sizes have to be much smaller than the local mean free path for a meaningful simulation. To obtain a better spatial and physical resolution, several mesh-refining strategies have been developed in the DSMC community. For Cartesian structured grids, a two-level mesh-refining approach has often been adopted [1,3]. One of the major advantages of this mesh-refining approach is the fast particle tracking as the background mesh. However, the mesh refining process may become awkward along the complex geometry of objects. For unstructured grids, an isotropic mesh-refining method based on h-refinement concept has been proposed [4,5]. One of the major advantages of this approach is the capability in body-fitting any complicated geometry of objects. However, there exists several major disadvantages, which include: 1) difficulty in maintaining mesh quality even with complicated mesh quality control algorithm, 2) very complicate hanging-node removing algorithm, especially for three-dimensional case [4] and 3) inefficient particle tracing on the refined cells. Thus, an alternative algorithm of mesh refinement on unstructured grids, which is free of the above problems, is critical in applying unstructured grids in the parallel DSMC method [6] .

Recently, Tseng *et al.* [7] had proposed a transient adaptive sub-cell (TAS) approach in DSMC utilizing unstructured grids to ensure better collision quality. In this paper, a new mesh-refining process for DSMC using unstructured grids, which is based on the idea of TAS, is introduced to "virtually" refine the background cells, which is named as virtual mesh refinement (VMR) algorithm.

2. VIRTUAL MESH REFINEMENT ALGORITHM

In general, the DSMC procedure involves: 1) moving the particles ballistically over a small time step and applying boundary conditions to particles which collide with boundaries, 2) indexing the particles within the grid of collision cells, 3) selecting particles from within the cells on a probabilistic basis and applying the collision routines to these, and 4) sampling the macroscopic flow properties from the collision cells. The cells are used to collision and sampling. To maintain a good quality of collision, the cell size has to be 1/2-1/3 of local mean free path, which is difficult in practice, since solution is generally not known *in priori*. In this section, we introduce a new mesh-refining procedure for the DSMC method which utilizes unstructured grids. This procedure is termed as two-level virtual mesh refinement (VMR), which is introduced next. It is termed "two-level" due to the fact that background grids are the first level while the refined grids are the second level.

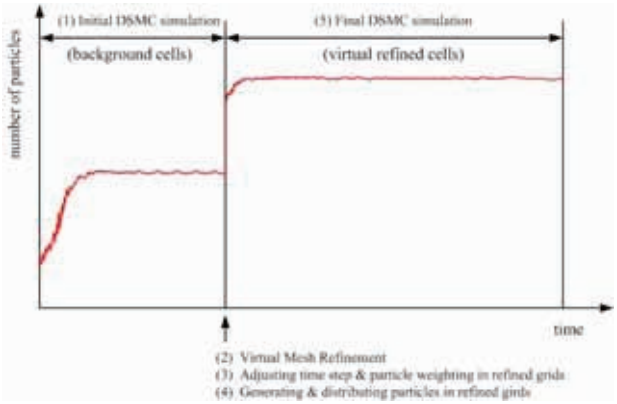


Fig. 1: *Temporal evolution of the proposed DSMC procedures with VMR.*

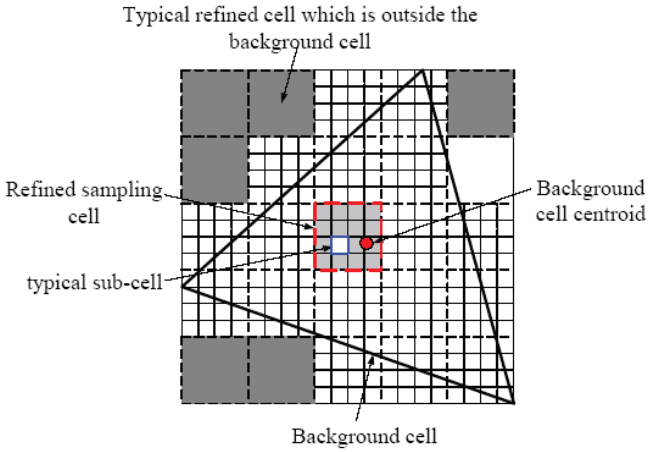


Fig. 2: *Typical refined cells (dashed lines) on a triangular background cell (solid lines) along with TAS.*

Fig. 1 shows the temporal evolution of the DSMC method with the VMR module, which is described next. These steps include: **1**) The initial DSMC simulation on the background grids, **2**) Virtual mesh refinement based on the data obtained in Step 1), **3**) Adjusting the time step size and particle weighting in the refined cells accordingly, **4**) Generating and randomly distributing particles in the refined cells based on Maxwellian distribution of velocities, and **5**) Final DSMC simulation on the refined grids. Note TAS function is used throughout the whole procedures to ensure the good collision quality. Some of the details in the above procedures are described in the following.

In Step **2**), the results of the initial DSMC simulation are used to determine the local mean free path in each background cell, which is then compared with the corresponding cell size. The result of comparison is then used to calculate the number of refined cells in each coordinate direction required to resolve the local mean free path in background cell. Note the refined cell is organized as Cartesian structured grids with the same cell size. Refined cell size is normally taken to be less than one half of the local mean free path, although it can be controlled by the user. A typical example is schematically shown in Fig. 2. One important advantage is the particle tracing becomes very efficient which results directly from the use of Cartesian structured grids for the refined cells. The sub-cell in each

background cell, which contains the background cell centroid, is also identified in this step. This will be used in the final data output. In addition, area of each sub-cell ("area" in two-dimensional case; "volume" in three-dimensional case), which is geometrically inside the background cell, is calculated using the Monte Carlo (MC) method. Note the area of the sub-cell (or volume) is required in calculating the number of collision pairs such as NTC method [1]. The reason not to apply the conventional method such as coordinate geometry is that it becomes very cumbersome and complicated as it is extended to three-dimensional case. The MC method is easy in concept as well as practical implementation, as shown schematically in Fig. 3. Each particle with randomly assigned position is checked if it is located in the background cell. Once it is located in the background cell, then the sub-cell which contains the particle is easily determined by taking advantage of the Cartesian structured sub-cells. Only those particles located inside the sub-cell and background cell are counted for the area calculation. The area of the i^{th} sub-cell inside the background cell is thus calculated as follows:

$$V_{vc_i} = V_c \times R_i / \sum_{i=1}^{N_{vc}} R_i \tag{1}$$

where R_i is the number of particles located inside the i^{th} sub-cell, V_c is the area of background cell and N_{vc} is the total number of sub-cells. Our experience shows that approximately 5,000*N_{vc} particles are required to reach 0.1% error for area calculations of all the sub-cells, which takes about 12.5 minutes of computational time for ~300,000 virtual sub-cells using 12 processors. This computational overhead is comparatively low as compared to the total DSMC simulation in general.

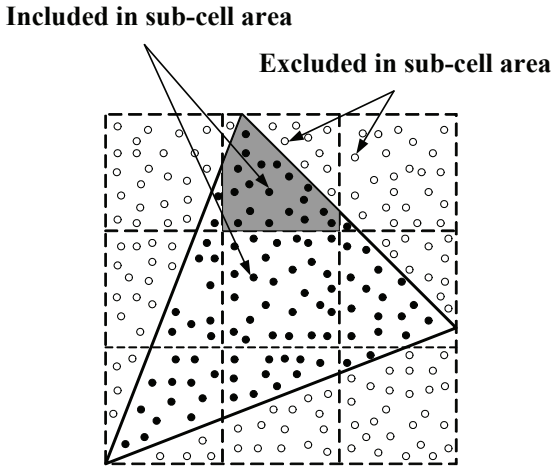


Fig. 3: *Sketch of calculating the sub-cell area inside an unstructured background cell using Monte Carlo method.*

3. RESULTS AND DISCUSSION

In this paper, we implement and verify the proposed VMR module in the PDSC by simulating a two-dimensional hypersonic flow over a circular cylinder (M=10) using various types of grids, including purely quadrilateral and mixed triangular and quadrilateral. Approximately 40 particles per cell for all the cases are maintained throughout the simulations. In addition, VHS collision model [1] and variable time step (VTS) scheme [4] are applied in the simulations unless otherwise specified. Note no chemical reactions are considered in the simulations. Finally, we apply this newly developed VMR algorithm in simulating two scramjet flows (M=12, Kn=0.02 and 0.06) using PDSC to demonstrate its capability in resolving flows with highly varying density.

3.1 M-10 Hypersonic Flow Past a Circular Cylinder

Note this problem has been adopted by Bird [8] as the benchmark for DSMC recently. Important free-stream flow conditions include: argon gas, velocity of 2634.1 m/s, temperature of 200K, number density of 4.274E20 m^{-3}, and Mach number of 10. Corresponding free-stream Knudsen number is 0.0091 based on the free-stream mean free path

(λ_∞=0.003 m) and the diameter of the cylinder (D=0.3048 m). There are four simulations, which are termed as Benchmark (only quadrilateral grids), VMR (mixed grids), TAS (mixed grids), and None (mixed grids). Note the cell size is 1/5-1/2 λ_∞ for the case of Benchmark using quadrilateral mesh, while those are only approximately 1-2 λ_∞ for the other three cases using mixed quadrilateral-triangular mesh. The number of cells of the former case is 195,000, while that of the latter cases is 12,825. Approximately 40 particles per cell are maintained for all the four simulation cases. Using 16 processors of a cluster, the computational time is ~18 hours for the benchmark case, while it is reduced to 5 hours for the Case-VMR, which can still faithfully reproduce the results of Case-Benchmark.

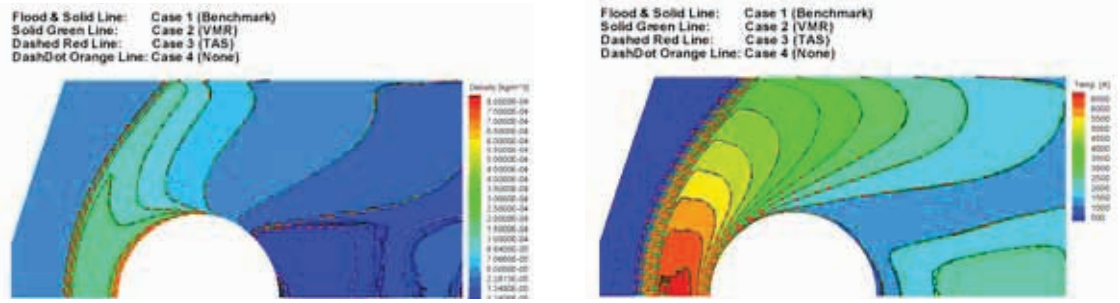


Fig. 4 *Contours of properties of Mach 10 hypersonic flow past a circular cylinder. (left) density. (right) temperature.*

Fig. 4 shows the contours of density and temperature for all the cases. Results clearly show that the results of the cases of VMR and TAS are very close to the benchmark data in general, while the results of Case-None shows large discrepancy with the Case-Benchmark, especially in the wake region. Note in the wake region the case-VMR is closer to the Case-Benchmark than the other two cases. Fig. 5 shows the surface properties as a function of angle measured from the front stagnation point. It shows that the local pressure coefficients are generally one order of magnitude larger than the local friction coefficients in most regions, except in the wake region. Results of pressure coefficient of all the test cases are in excellent agreement with those of the benchmark case. The friction coefficients of Case-None and Case-TAS clearly over predict the peak value, while those of Case-VMR agree very well with those of Case-Benchmark. Similarly, the heat transfer coefficients of Case-None and Case-TAS deviate very much from those of Case-VMR and Case-Benchmark.

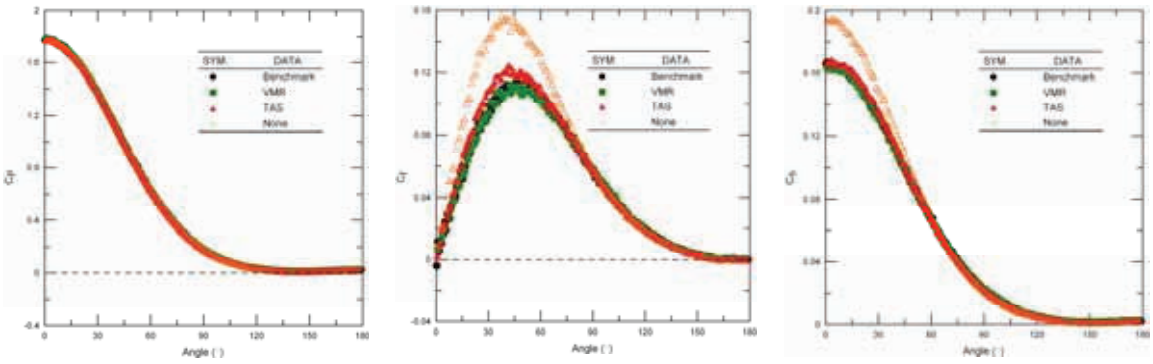


Fig. 5 *Surface property distribution as a function of distance from the front stagnation point. (left) Local pressure coefficient. (middle) local friction coefficient. (right) local heat transfer coefficient.*

3.2 M-12 Scramjet Flow

Fig. 6 illustrates the sketch of the simulation domain of the scramjet flows. Important free-stream flow conditions include: argon gas, velocity of 2634.1 m/s, temperature of 208K, number densities of 4.23E+21 (Kn=0.02) and 1.41E+21m^{-3} (Kn=0.06), and Mach number of 12. Number of particles is approximately 30 and 10 millions, respectively, for the Kn=0.02 and Kn=0.06 case using 64 processors of IBM-1390 at National Center for High-Performance Computing of Taiwan.

Fig. 7 shows the simulated pressure and temperature contours of the two cases. Results show that contour becomes smoothed out as rarefaction increases. Results of surface properties and paralel performance will be presented in the conference

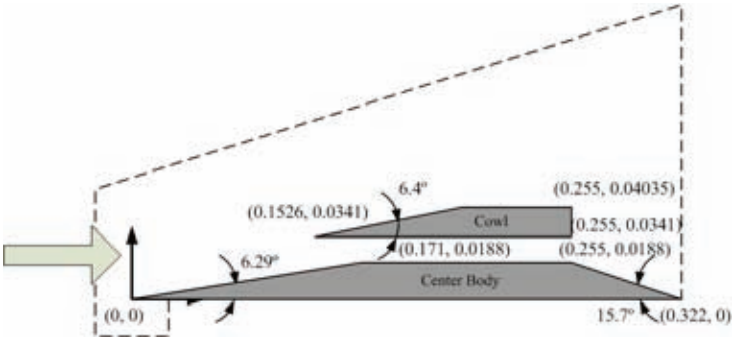


Fig. 6 *Sketch of the simulation domain of a scramjet flow (unit: meter)*

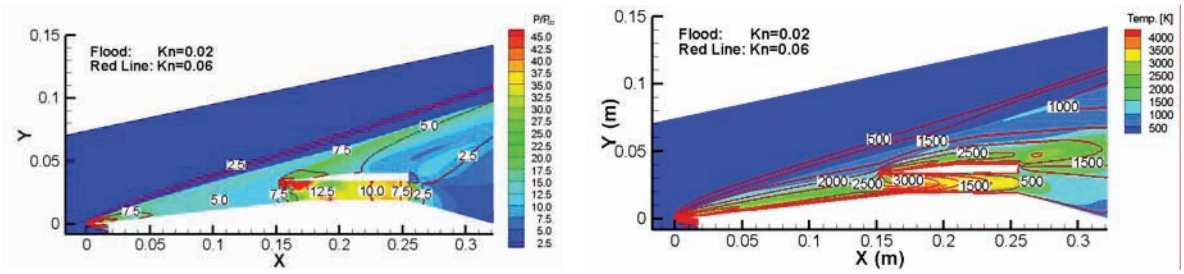


Fig. 7 *Simulated pressure (left) and temperature(right contours of the scramjet argon flows(M=12).*

4. CONCLUSIONS

In this paper, we have developed a virtual mesh refinement (VMR) and implemented in the parallel direct simulation Monte Carlo code (PDSC) which utilizes unstructured grids. A M-10 argon flow past a cylinder was used to demonstrate the use of VMR not only can reproduce the results of time-consuming benchmark case, but also can greatly reduce the computational time. Finally, a 2D scramjet argon flow was simulated to demonstrate the robustness of this new mesh-refining algorithm in unstructured DSMC.

REFERENCES

1. Bird, G.A, (1994). *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, Oxford University Press.
2. Wagner, W. (1992). *J. Stat. Phys.* **66**, 1011-1044.
3. LeBeau, G.J. (1999). *Comput. Meth. Appl. Mechanics Eng.*, **174**, 319-337.
4. Wu, J.-S., Tseng, K.-C. and Wu, F.-Y. (2004). *Comput. Phys. Comm.*, **162**, 166-187.
5. L. Wang L and J.K. Harvey (1994). *Rarefied Gas Dynamics*, Harvey J, Lord G (eds). 19th International Symposium on Rarefied Gas Dynamics, 843-849.
6. Wu, J.-S. and Tseng, K.-C. (2005). *Int. J. Num. Meth. Eng.*, **63**, 37-76.
7. Tseng, K.C. , Cave, H.M., Lian, Y.-Y., Kuo, T.-C., Jermy, J.C. and Wu, J.-S. (2008). *Computer & Fluids* (submitted).
8. Bird, G.A. (2006). "Sophisticated Versus Simple DSMC," *25th International Symposium on Rarefied Gas Dynamics*, St. Petersburg, Russia.

Numerical simulation of scattering of helioseismic MHD waves by sunspots

Parchevsky, K.V., Kosovichev, A.G.

(e-mail: kparchevsky@solar.stanford.edu)
(e-mail: sasha@sun.stanford.edu)
Stanford University, HEPL, Stanford CA 94305, USA

Abstract: We present results of numerical 3D simulation of interaction of MHD waves with the sunspot. Self consistent magnetohydrostatic model of the sunspot was chosen as the background model. Using filtering technique we separated magnetoacoustic and magnetogravity waves. It is shown, that inside the sunspot magnetoacoustic and magnetogravity waves are not spatially separated unlike the case of the horizontally uniform background model. The sunspot causes anisotropy of the amplitude distribution along the wavefront and changes the shape of the wavefront. The amplitude of the waves is reduced inside the sunspot. This effect is stronger for the magnetogravity waves than for magnetoacoustic waves. The shape of the wavefront of the magnetogravity waves is distorted stronger as well.

Keywords: solar oscillations, sunspots, MHD waves

1. INTRODUCTION

Turbulent convection in the Sun generates acoustic waves. These waves can be observed at the surface by measuring the doppler velocities. Using techniques of local helioseismology it is possible to reconstruct the internal structure (vertical profiles of sound speed and flows) of active regions in the Sun from such observations. In the background magnetic field of active regions different types of waves can exist: fast, slow, Alfven, and magnetogravity waves. All these waves have different dispersion relations, velocities and can convert to each other, that even more complicates the analysis.

In general, the main factors causing variations in wave amplitude and helioseismic travel times in solar magnetic regions, can be divided in two types: direct (due to the additional magnetic restoring force) and indirect (due to the changes in the convective and thermodynamic properties in magnetic regions). The indirect effects of the suppressed excitation of acoustic waves in sunspot regions were investigated in [1]. The issue of the influence of the inclined magnetic field was raised by Schunker et al. in [2], who found that the phase shift of the signal in the penumbra of a sunspot, measured by the acoustic holography technique varies with the sunspot position on the disk. They attributed this to the variations of the angle between the inclined magnetic field of the penumbra

and the line-of-sight. In this situation the detailed numerical simulation of scattering of MHD waves by sunspots is extremely welcome.

2. GOVERNING EQUATIONS AND NUMERICAL SCHEME

Propagation of MHD waves inside the Sun is described by the following system of linearized equations:

$$\begin{aligned}
&\frac{\partial \rho'}{\partial t} + \nabla \cdot \boldsymbol{m}' = 0, \\
&\frac{\partial \boldsymbol{m}'}{\partial t} + \nabla p' - \frac{1}{4\pi}\left[(\nabla \times \boldsymbol{B}') \times \boldsymbol{B}_0 + (\nabla \times \boldsymbol{B}_0) \times \boldsymbol{B}'\right] = \rho' \boldsymbol{g}_0 + \boldsymbol{S}, \\
&\frac{\partial \boldsymbol{B}'}{\partial t} = \nabla \times \left(\frac{\boldsymbol{m}'}{\rho_0} \times \boldsymbol{B}_0\right), \\
&\frac{\partial p'}{\partial t} + c_{s0}^2 \left[\nabla \cdot \boldsymbol{m}' + \boldsymbol{m}' \cdot \left(\frac{\nabla p_0}{\Gamma_1 p_0} - \frac{\nabla \rho_0}{\rho_0}\right)\right] = 0,
\end{aligned} \tag{1}$$

where $\boldsymbol{m}' = \rho_0 \boldsymbol{v}'$ is the momentum perturbation, $\boldsymbol{v}'$, ρ', p', and $\boldsymbol{B}'$ are the perturbations of velocity, density, pressure, and magnetic field respectively, $\boldsymbol{S} = (0, 0, f(x, y, z, t))^T$ is the wave source function, $f(x, y, z, t)$ is the density of z-component of force, ρ_0, p_0, c_{s0}, and $\boldsymbol{B}_0$ are the density, pressure, sound speed, and magnetic field in the background state.

The spatial and temporal behavior of the wave source is modeled by function $f(x, y, z, t) \equiv AH(\boldsymbol{r})F(t)$:

$$H(x, y, z) = \begin{cases} \left(1 - \dfrac{r^2}{R_{src}^2}\right)^2 & \text{if } r \le R_{src} \\ 0 & \text{if } r > R_{src} \end{cases} \tag{2}$$

$$F(t) = \left(1 - 2\tau^2\right) e^{-\tau^2}, \quad \tau = \frac{\omega_0 (t - t_0)}{2} - \pi, \quad t_0 \le t \le t_0 + \frac{4\pi}{\omega_0}, \tag{3}$$

where A is the amplitude of the source, R_{src} is the source radius, r is the distance from the source center, ω_0 is the central source frequency, t_0 is the moment of the source initiation. This source model provides the wave spectrum, which closely resembles the solar spectrum. It has a peak near the central frequency ω_0 and spreads over a broad frequency interval.

We developed a semidiscrete code of high order for numerical solution of equations (1). The spatial high-order finite difference (FD) scheme was optimized to preserve the dispersion relation of the continuous problem (see [3]). The coefficients of this FD scheme are chosen from the requirement that the error of the Fourier transform of the spatial derivative is minimal. It can be shown that the 4th-order dispersion relation preserving (DRP) FD scheme describes short waves more accurately than the classic 6th-order FD scheme. A 3rd-order, three-stage strong stability preserving (SSP) Runge-Kutta scheme, described in [4], with the Courant number, $C = 1$, is used as a time advancing scheme.

To prevent spurious reflections of acoustic waves from the horizontal boundaries, we established non-reflecting boundary conditions based on the Perfectly Matched Layer (PML) method at the top and bottom boundaries. The method is described in [5]. The top boundary was set at the height of 500 km above the photosphere. This simulates a realistic situation when not all waves are reflected from the photosphere. Waves with frequencies higher than the acoustic cut-off frequency pass through the photosphere and are absorbed by the top boundary. This naturally introduces frequency dependence of the reflecting coefficient of the top boundary. The lateral boundary conditions are periodic.

The efficiency of the high-order FD schemes can be reached only if they are combined with adequate numerical boundary conditions. We followed Carpenter et al. [6] and used an implicit Padé approximation of the spatial derivatives near the top and bottom boundaries to derive a stable 3rd-order numerical boundary conditions consistent with the 4th-order DRP numerical scheme for interior points of the computational domain. Waves with the wavelength less than $4\Delta x$ are not resolved by the FD scheme. They lead to point-to-point oscillations of the solution that can cause a numerical instability. Such waves have to be filtered out. We use a 10th-order digital filter to eliminate unresolved short wave component from the solution at every 5th time step. Details of numerical realization of the code can be found in [7]. The code is written in C++, parallelized, and optimized for running on multiprocessor super computing systems with shared memory.

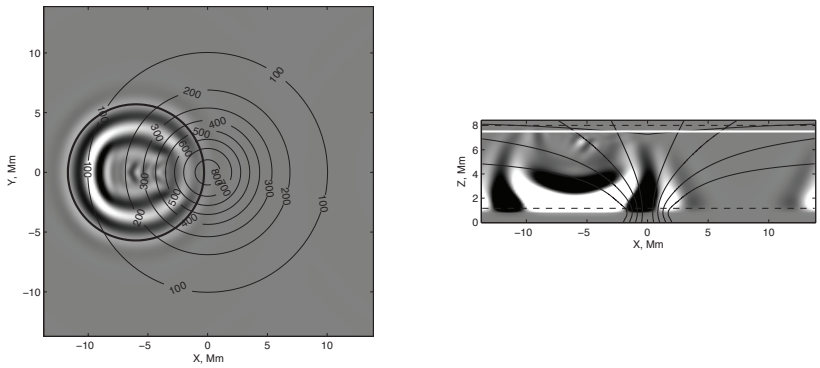


Fig. 1.— *Snapshot of the z-component of momentum $\rho_0 V_z'$ at moment t=16.7 min. The left panel represents the horizontal slice of the domain at the photospheric level. The right panel represents the vertical slice. Solid black curves in the right panel represent magnetic filed lines.*

3. RESULTS AND DISCUSSION

In this section we present our results of numerical simulation of propagation of MHD waves through the sunspot in Cartesian geometry. The domain of simulation is a box of size 27.6×27.6×7.5 Mm^3 (184×184×62 nodes). Waves are generated by a single source of vertical force with central

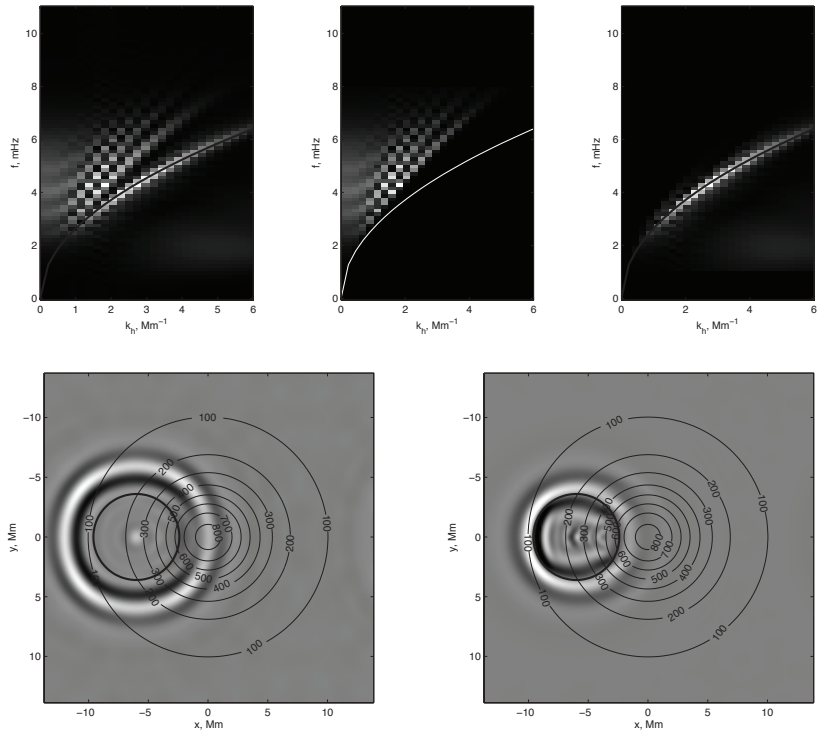


Fig. 2.— *Separation of p- and f-modes by filtering. The top row represents the original and filtered k-ν diagrams. The bottom row represents the corresponding maps of z-component of velocity at the moment of t = 20 min. for p- and f-modes respectively. The solid circle marks the inner part of the p-mode wavefront. The black and white solid curves in the panels of the top row show the position of the theoretical f-mode ridge.*

frequency $\nu = 3.5$ mHz placed 100 km below the photosphere. The horizontal grid is uniform with $\Delta x = \Delta y = 150$ km. The vertical grid is non-uniform. The grid step Δz varies from 50 km near the photosphere to 400 km near the bottom of the computational domain. Time step $\Delta t = 0.1$ s was chosen to satisfy the Courant stability condition. The background model of the sunspot is based on solution of the nonlinear magnetostatic equations (for details see [8]). The values of the magnetic field at the axis of the sunspot are 0.843 kG and 28.9 kG at the photospheric level and the depth of 6.7 Mm respectively.

The simulation results are shown in Figure 1. The horizontal white line in the left panel near the top boundary marks the position of the photosphere. The generated wave field is a mixture of fast, slow MHD, Alfven and surface magnetogravity waves. The Alfven wave is generated at the source location and slowly propagates inside the sunspot along the magnetic field lines. Magnetogravity and magnetoacoustic waves have different speed, dispersion relations and can be easily separated by applying an appropriate filter in k-ω space. The result of such separation is shown in Figure 2. The

bottom row represents horizontal maps of V_z' at the photospheric level for p-modes (left panel) and f-modes (right panel). These maps are obtained by the inverse Fourier transform of corresponded filtered k-ω diagrams from the top row. The black solid circle in the panels of the bottom row marks the inner part of the p-mode wavefront. It is clear, that unlike the case of horizontally uniform background model, magnetoacoustic and magnetogravity modes are not spatially separated inside sunspots.

The non-uniform background model causes anisotropy of the amplitude distribution along the wavefront and asymmetry of the wavefront itself. When the wave enters the sunspot the part of the wavefront closest to the sunspot center speeds up that deforms the shape of the wavefront. This effect is stronger for the magnetogravity modes than for for magnetoacoustic modes. Amplitude of the wave is smaller inside the sunspot. Decreasing of the wave amplitude for density and velocity variations near the center of the sunspot is caused by two reasons: variations of the background density and transformation into different types of MHD waves. When the wavefront of the magnetoacoustic mode passes the sunspot, its amplitude is restored to the previous value while the part of the wave front of the magnetogravity wave which passed through the sunspot shows deficit of the amplitude. This can be an evidence that the magnetogravity wave is transformed into different types of MHD waves more than the magnetoacoustic wave.

4. ACKNOWLEDGEMENTS

We thank to Elena Khomenko for providing us self-consistent magnetohydrostatic model of the sunspot which we used as the background model in our simulations. We thank to NASA Ames Research Center for providing the opportunity to use Columbia supercomputer for our simulations.

REFERENCES

1. Parchevsky, K. V., & Kosovichev, A. G. 2007b, ApJ, 666, L53

2. Schunker, H., Braun, D. C., Cally, P. S., & Lindsey, C. 2005, ApJ, 621, L149

3. Tam, C. and Webb, J. 1993, J. Comput. Phys. 107, 262

4. Shu, C.-W. 2002, in Collected Lectures on the Preservation of Stability under Discretization, eds. D. Estep and S. Tavener, SIAM, 51.

5. Hu, F. Q. 1996, J. Comput. Phys., 129, 201.

6. Carpenter, M.H., Gottlieb, D., and Abarbanel, S. 1993, J. Comput. Phys. 108, 272.

7. Parchevsky, K. V. & Kosovichev, A. G. 2007a, ApJ, 666, 547

8. Pizzo, V. J. 1986, ApJ, 302, 785.

POSTER ABSTRACTS

The Design Methodology and Numerical Modelling of a Vertical Axis Wind Turbine.

Joy Pathak
University of Windsor, Windsor,ON, N9B2P4, Canada
(Tel: 519-969-9625; e-mail: pathak5@uwindsor.ca)

Abstract: In this present age as Computational Fluid Dynamics (CFD) grows, there are immense opportunities of using CFD tools in standard undergraduate curriculum. The present investigation is aimed at exploring the power and efficiency of a two blade Savonius Wind turbine. Computational Fluid Dynamics software package (Fluent) was utilized to make these comparisons with extremely well refined meshes and theoretically found boundary conditions and parameters. Performance analysis has been on the basis of starting characteristics, static torque and rotational speed. The major aim of this study is to go through the development and comparisons to create a Numerical Model of 2D Vertical Axis Wind Turbine (VAWT) that can be used as a template for any VAWT aerodynamic analysis.

The author has investigated two major meshing techniques to create the Numerical Model.The first part models a steady state and unsteady Deformation meshing technique, whereas the second part uses a Sliding Mesh model.

The paper also provides guidance to undergraduate students and upcoming researchers in the area of Computational Fluid Dynamics on how to create 2D meshes of such applications in Gambit and the methodology and process required to further solve it in an appropriate solver

Keywords: Wind turbines, Savonius, Vertical Axis, Sliding Mesh

1. INTRODUCTION

Wind power is the conversion of wind energy into a useful form, such as electricity, using wind turbines. At the end of 2008, worldwide nameplate capacity of wind-powered generators was 120.8 gigawatts[3]. Although wind produces only about 1.5% of worldwide electricity use [3] it is growing rapidly, having doubled in the three years between 2005 and 2008. One of the wind rotors of a vertical axis, Savonius wind rotor was developed by a Finnish engineer, Sigurd Savonius, in 1925 [14]. The performance of these rotors is lower than that of the other conventional wind rotors but even so, they have a number of advantages over the others. For example, design of such rotors is simple and cheap. They start to run on their own and they are independent of the direction of the wind. They also have a high starting torque. Despite such a certain number of advantages of Savonius wind rotors; they are not preferred so much due to their low aerodynamic performance levels. To eliminate this disadvantageous quality of Savonius wind rotors, several studies have been done in recent years in order to improve their aerodynamic performance. Therefore, a lot of theoretical and experimental studies have been carried out to increase the performance of Savonius wind rotors. A number of scientists have tested many models through the studies that they have done in the static and dynamic state of Savonius wind rotor. In these studies they have experimentally and numerically examined the effects of various design parameters such as the rotor aspect ratio, the overlap and the separation gap between rotor buckets, the profile change of the bucket cross-section, the number of buckets, the presence or absence of rotor endplates, and the influence of bucket stacking [14–19]. Significant steps have been taken in the improvement of power and torque performances of Savonius wind rotors through the above mentioned studies. In another group of studies, however, pressure distributions on the blades have been measured to analyze the flow field and aerodynamic performance in and around a rotating and static Savonius wind rotor experimentally and numerically [20–31]. Aldoss [26] has carried out an experimental study on the aerodynamic performance of Savonius wind rotor by using a swinging blade rotor. Later, Aldoss et al. [27] have developed this study and improved the performance of Savonius wind rotor by allowing the rotor blades to swing back with an optimum angle. Tabassum and Probert [28] and Reupke and Probert [29] have found out that with the Savonius wind rotor

with hinged blades, which they have separately designed, higher torques could be acquired at rather lower end speeds than with the conventional Savonius wind rotors.

1.1 Present Study

A conventional Savonius wind rotor is made up of two semi-cylinders, called blades, which are placed in between two horizontal discs and the centers of which are symmetrically sided. The wind hitting the Savonius wind rotor at a certain speed creates a positive torque in the inner part of the cylinder forming the rotor and a negative torque in its outer part. Since the torque in its inner part is higher than the torque in the outer part, a rotation movement is secured. Below is a simple figure for the running principle of the Savonius wind rotor (Fig 1). In the present study, Fluent 6.3 Student Version software has been used as the computational fluid dynamics (CFD) package program. The Savonius wind rotor in the dynamic position has been analyzed from aerodynamic aspects. Fluent is a CFD software using the finite sizes method and has the capacity of solving the flow around complex geometries. Modelling of the study has been made by using the Gambit 2.0, a program of Fluent for creating a model and network. In the Fluent 6.3 program, physical properties of the flow have been defined; analysis technique and turbulence model have been selected; the number of iterations and convergence values have been determined by entering the values of boundary conditions, and finally analysis have been made. To do the numerical analysis in a shorter time, the model has been formed to have two dimensions. The information determined is compared the available experimental such as the results of Burçin Deda Altan and Mehmet Atılgan (9).

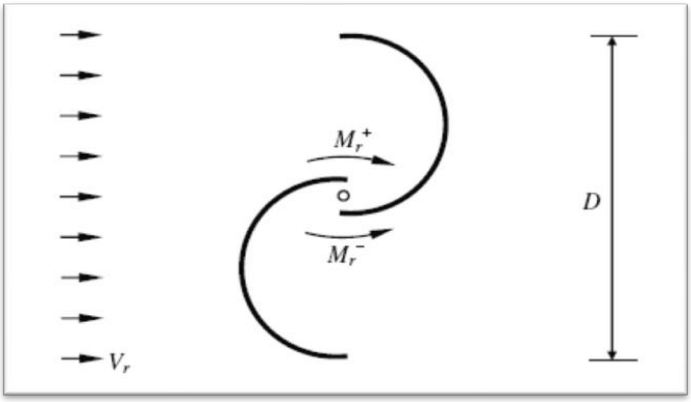


Fig 1: Direction of the Torque exerted by the wind on the turbine blades.

2.1 Steady flow

One of the first steps in steady-flow analysis is to locate control points; that is, points along the stream where the elevation can be computed once the steady flow is selected. At least one point of known elevation is needed to start the computations. For subcritical flow, this point, called an initial condition, will be the downstream boundary of the region of interest.

In Fluent 6.3, the flow features associated with multiple rotating parts can be analyzed using the multiple reference frame (MRF) capability. This model is powerful in that multiple rotating reference frames can be included in a single domain. The resulting flow field is representative of a snapshot of the transient flow field in which the rotating parts are moving. However, in many cases the interface can be chosen in such a way that the flow field at this location is independent of the orientation of the moving parts. In other words, if an interface can be drawn on which there is little or no angular dependence, the model can be a reliable tool for simulating time-averaged flow fields. It is therefore very useful in complicated situations where one or more rotating parts are present. [25]

2.2 Standard K-epsilon Turbulence model

The K-epsilon model is one of the most common two equation turbulence models.

2.3 Results

Angular Speed (rad/s)	Total force (N)	Torque(n.m)	Power	Power Factor
3	4.76	2.82	8.57	0.31
5	4.88	2.89	14.30	0.52
7	4.98	2.97	19.31	0.71
9	5.02	3.01	27.13	1.02
11	5.03	3.06	33.20	1.22

Table 1: Parameters obtained from Fluent 6.3

2.4 Discussion

The K-epsilon model predicts excessive levels of turbulence shear stress, particularly in the presence of adverse pressure gradients leading to suppression of separation on curved walls. The values in Table 1 indicate that the power factor is increasing in a linear manner. According to the experimental results used to compare the computational results, the maximum power factor achieved should be below 11rad/s. This therefore proves that this numerical model is not the optimum simulation of real life characteristics of a Vertical Axis Wind Turbine.

3.1 Unsteady

In unsteady-flow analysis, computational elements and algebraic approximations to the differential or integral terms in the governing equations must be used to develop two algebraic equations for each computational element written in terms of elevations and flows at the ends of the element. These governing equations are more complex than those for steady-flow analysis. For unsteady flow, a computational element with respect to time also must be considered, but it is simple: the time axis is divided into finite increments that, ideally, will be short enough so that the algebraic approximations of the differential and integral terms will be sufficiently accurate. Because of this dependence on time, the algebraic governing equations involve not only the unknown flow and elevation at two points along the channel but also at two points in time.

3.2 K epsilon

The turbulence kinetic energy and the turbulent dissipation rate were set as 1. The inlet velocity was chosen as 5 m/s. The angular velocity of the blades was increased in 2 rad/s increments starting at 3 rad/s. The mesh was adapted with velocity gradients after initial iterations of Time step size 0.002 seconds and 20 iterations per time step.

Angular Speed (rad/s)	Total force (N)	Torque(n.m)	Power (kWatts)	Power Factor
3	11.82	7.09	0.02	0.07
5	14.17	8.50	0.04	0.15
7	14.94	8.96	0.06	0.22
9	19.54	11.72	0.10	0.37
11	16.33	9.79	0.09	0.29

Table 2: Parameters obtained from Fluent 6.3

3.3 Discussion

Due to computational time and resources only one model of unsteady solution through the MRF method was considered. After viewing Table 2 it can be noted that the Power factor maxes out around approximately 9 rad/s. Comparing the power factor results to the experimental results conducted by previous researchers the values are from the Power Factor vs. Angular velocity curves obtained by them. The results obtained from the unsteady k-epsilon model are the closest to experimental results compared to the previous methods.

4.1 Sliding Mesh Model

For the second half of the study, a sliding mesh model was created. Two separate meshe were created. These were then merged with the 'tmerge' capabilities of Fluent 6.3. The model was then exported as a 2D mesh and imported into Fluent 6.3 for solving. The parameters of the Unsteady K-epsilon model previously used was against implemented for the sliding mesh model.

4.2 Results

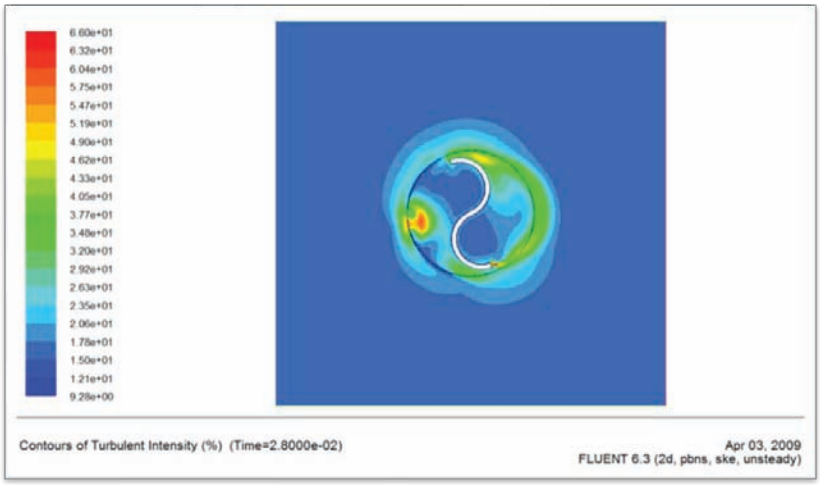


Fig 2: Turbulence Intensity Contours – 10rad/s – Sliding Mesh Model

Angular Speed (rad/s)	Total force (N)	Torque(n.m)	Power (kWatts)	Power Factor
3	12.24	7.34	22.03	0.08
5	23.72	14.22	71.16	0.263556
7	18.71	11.22	78.58	0.29
9	11.52	6.91	62.20	0.23
11	8.22	4.93	54.25	0.20

Table 3: Parameters obtained from Fluent 6.3

4.3 Discussion

As can be viewed from Figure 2 the rotation of the meshes has been accomplished. This is unlike the previous method where the fluid was modeled to rotate using the reference frame technique.

Table 3 clearly shows that the maximum power factor is approximately around 5 and 7 rad /s. These results tally with the experimental results conducted by previous researchers [11] proving that this is the most optimum Numerical model for validating Vertical Axis Wind Turbines.For future, relaxation schemes and different discretization schemes might be investigated to increase the convergence rate.

References

1) Joselin Herbert GM, et al. A review of wind energy technologies. Renewable and sustainable energy reviews. *Renew Sustainable Energy Rev* 2007; 11:1117–45.
2) Global wind energy market as of 2006. *Press release of Global Wind Energy Council* (GWEC).
3) World Wind Energy Association (06-Feb-2009). *"120 Gigawatt of wind turbines globally contribute to secure electricity generation".* Press release.
4) Ushiyama I, Nagai H, Mino M. The optimum design configurations of Savonius wind turbines. In: *Proceedings of 17th intersociety energy conversion engineering conference,* 1982. p. 2096–101.
5) Saylers AT. Blade configuration optimization and performance characteristics of a simple Savonius rotor. *In: Proceedings of institution of mechanical engineers*, vol. 199, 1995. p. 185–91.
6) Sheldahl RE, Blackwell BF, Feltz LV. Wind tunnel performance data for two and three bucket Savonius rotor. *Journal of Energy* 1978;2:160–4.
7) Modi VJ, Fernando MSUK. On the performance of the Savonius wind turbine. *ASME Journal of Solar Energy Engineering 1989*;111:71–81.
8) Vishwakarma R. Savonius rotor wind turbine for water pumping—an alternate energy source for rural sites. *Journal of Institution of Engineers* (India) 1999;79:32–4.
9) Burçin Deda Altan *, Mehmet Atılgan, An experimental and numerical study on the improvement of the performance of Savonius wind rotor. *Energy and Conversion journal*. 49, 2008.
10) Alexander AJ, Holownia BP. Wind tunnel test on a Savonius rotor. *J Ind Aerodyn* 1978; 3:343–51.
11) Modi VJ, Fernando MSUK. On the performance of the Savonius wind turbine. *J Sol Energ Eng* 1989;111:71–81.
12) B Mohammadi, O Pironneau – *Analysis of the K-epsilon Turbulence Model*,1994 - John Wiley & Son Ltd
13) Menter, F. R. "Zonal Two Equation k-ω Turbulence Models for Aerodynamic Flows", AIAA, 1993 Paper 93-2906.
14) Ushiyama _I, Nagai H. Optimum design configurations and performance of Savonius rotors. *Wind Eng* 1988;12(1):59–75.
15) Sheldahl RE, Blackwell BF, Feltz LV. Wind tunnel performance data for two and three bucket Savonius rotors. *J Energy* 1978;2(3):160–4.
16) Alexander AJ, Holownia BP. Wind tunnel test on a Savonius rotor. *J Ind Aerodyn* 1978;3:343–51.
17) Modi VJ, Fernando MSUK. On the performance of the Savonius wind turbine. *J Sol Energ Eng* 1989;111:71–81.
18) Mojola OO, Onazanya OE. Performance testing of a Savonius windmill rotor in shear flows. *Wind Eng* 1984;8(2):109–21.
19) Mojola OO. On the aerodynamic design of the Savonius windmill rotor. *J Wind Eng Ind Aerodyn* 1985; 21:223–31.
20) Fujisawa N. Velocity measurements and numerical calculations of flow fields in and around Savonius rotors. *J Wind Eng Ind Aerodyn* 1996;59:39–50.
21) Fujisawa N. On the torque mechanism of Savonius; rotors. *J Wind Eng Ind Aerodyn* 1992;40:277–92.
22) Fujisawa N, Taguchi Y. Visualization and image processing of the flow in and around a Savonius rotor. *J Flow Visual Image Process* 1993;1:337–46.
23) Fujisawa N, Shirai H. Experimental investigation on the unsteady flow field around a Savonius rotor at the maximum power performance. *Wind Eng* 1987;11(4):195–206.
24) Fujisawa N, Gotoh F. Pressure measurements and flow visualization study of a Savonius rotor. *J Wind Eng Ind Aerodyn* 1992; 39:51–60.
25) Fujisawa N, Gotoh F. Visualization study of the flow in and around a Savonius rotor. *Exp Fluids* 1992;12:407–12.
26) Aldoss TK, Kotb MA. Aerodynamic loads on a stationary Savonius rotor. *JSME Int J* 1991;Series II, 34(1):52–5.
27) Aldoss TK, Kotb MA. Theoretical calculations of the flow field around a Savonius rotor. *Wind Eng* 1988; 12(3):194–203.
28) Ishimatsu K, Kage K, Okubayashi T. Numerical study for the flow fields and performances of Savonius-type and bach-type rotors. In: *The 10th international symposium on flow visualization*, Kyoto, Japan; 2002. p. s.1–7.
29) Kotb MA. Flowfield around a partially-blocked Savonius rotor, *Appl Energy* 1991;38:117–32.

Direct numerical simulation of Couette flows inside a square duct

Hsin-Wei Hsu[a]and Chao-An Lin[a*]

[a]Department of Power Mechanical Engineering, National Tsing Hua University, Hsinchu 30013, TAIWAN

1. Introduction

Turbulent Poiseuille or Couette flows inside a square or rectangular cross-sectional duct are of considerable engineering interest because of their relevance to the compact heat exchangers and gas turbine cooling systems. The most studied flow is the turbulent Poiseuille type inside a square duct. Experimental studies on turbulent Poiseuille flows had shown that, near the corners, a transverse circulatory flow exists which is not observed in circular ducts nor in laminar rectangular ducts. The transverse motion is derived from the anisotropy of the turbulent stresses and is identified as the secondary flow of Prandtl's second kind.

Numerical simulations of large eddy (LES) and direct numerical (DNS) simulations of square duct flows were conducted by Madabhushi and Vanka [1] and Gavrilakis [2], respectively, where the bulk Reynolds numbers are 5810 and 4410. Square duct flow but at higher bulk Reynolds number, 10320, were also investigated by Huser and Biringen [3] using DNS. There are also investigations directed to explore the influences of the bounding wall geometry, non-isothermal effect, free surface and system rotation on the secondary flow pattern within turbulent Poiseuille duct flows However, little is known about the effect of moving wall on the turbulence anisotropy and hence the resulting secondary flow within duct flows, and a brief report was documented in [4]. Therefore, the present study aims at obtaining a detailed description of turbulent Poiseuille flows and Couette flows inside square duct through the DNS. The framework of the present numerical procedure incorporates the finite volume method and the staggered grid arrangement. The time integration method is based on a semi-implicit, fractional step method. Flows considered here are fully developed flow within square duct and the geometry is shown in Figure 1. Grid (128x128x256) is symmetrically clustered using hyperbolic tangent functions towards the walls on the cross-plane of the duct with minimum and maximum spacing$\Delta\ x^+$,Δy^+ approximately as 0.2 and 5.94. In the streamwise direction, the grid is uniformly distributed with $\Delta z^+ \sim 8.9$.

2. Results

The basic flow parameters are summarized in Table 1, where PP stands for plane Poiseuille flow, PC is plane Couette flow, DP is square duct Poiseuille flow and DC is square duct Couette flow. Reynolds number based on the bulk velocity is ranging from 5400 to 6200. For PP, PC and DP, the stationary wall friction Reynolds number($Re_{\tau b}$) is around 360 based on the channel

*Email address: calin@pme.nthu.edu.tw

and duct height, and is 300 for DC. The effects of the boundary wall and the Couette velocity on the turbulence distributions can be observed from Figures 2 and 3. Here, the DNS data of plane channel flows (Moser et al.[5] and Abe et al.[6], $Re\tau = 180$) and plance Couette flow (Kawamura et al.[7] are also included for comparisons. It should be for square duct flows, the profiles are along wall bisector at the stationary wall. It can clear observed by reference to Fig. 2 that for the Poiseuille flow, the influences of the bounding wall is marginal. However for the Couette flow shown in Fig. 3, the turbulence intensities of the square duct flow is much higher than those in the plane channel flow, indicating the extra strains induced by the bounding wall. Figures 4 and 5 show the turbulent energy budgets for Poiseuille and Couette flow. Again, the effects of the boundary wall is negligible for the Poiseuille flow. However, for the DC flow, it can be clearly seen that the maximum production is 13 % higher than its plane counterpart. This excess of production is further enhanced at the duct center region. It should be also noted that at this region the level of pressure-strain exceeds that of the turbulence dissipation. However, in the plane Couette flow, the magnitudes of these two processes are compatible.

REFERENCES

1. R. K. Madabhushi and S. P. Vanka, Large eddy simulation of turbulence driven secondary flow in a square duct, Phys. Fluids A **3**, (1991) 2734-2745.
2. S. Gavrilakis, Numerical simulation of low-Reynolds-number turbulent flow through a straight square duct, J. Fluid Mech. **244**, (1992) 101-129.
3. A. Huser and S. Biringen, Direct numerical simulation of turbulent flow in a square duct, J. Fluid Mech. **257** (1993) 65-95.
4. W. Lo & C. A. Lin, 2006, Mean and turbulence structures of Couette-Poiseuille flows at different mean shear rate in a square duct. Phys. Fluid **18**,068103.
5. R. Moser, J. Kim & N. Mansour, 1999, Direct numerical simulation of turbulent channel flow up to Retau=590. Phy. Fluids **11**, 943.
6. H. Abe, H. Kawamura & Y. Matsuo, 2001, Direct numerical simulation of a fully developed turbulent channel flow with respect to the Reynolds number dependence. J. of Fluids Engineering-Transacations of ASME **123**, 382-393.
7. H. Kawamura, H. Abe & K. Shongai, 2000, DNS of turbulent and heat transport in a channel flow with different Reynolds and Prandtl numbers and boundary conditions. 3rd In. Symp. on Turbulence, Heat and Mass Transfer.

Table 1
TABLE 1. The flow conditions for simulated cases; W_w denotes the velocity of the moving wall and W_{Bulk} is the bulk velocity; $Re_c = \frac{W_w D}{\upsilon}$

	Re_{Bulk}	$Re_{\tau b}$	Re_c
Case PP	6244	356	0
Case PC	5628	362	12800
Case DP	5477	357	0
Case DD	5426	295	17130

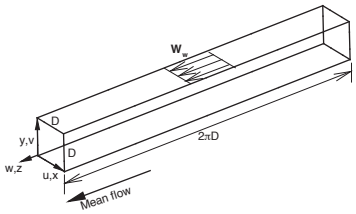


Figure 1. Computational domain and Mean streamwise velocity along the wall-bisector

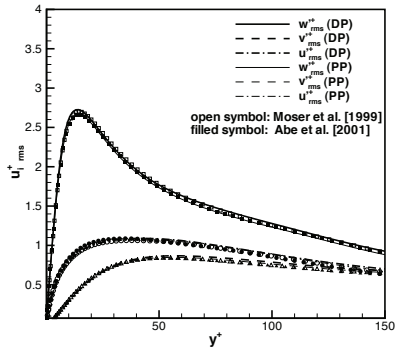


Figure 2. Turbulence intensity-Poiseuille flow along the wall bisector

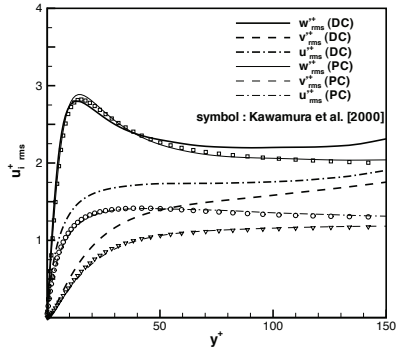


Figure 3. Turbulence intensity-Couette flow along the wall bisector

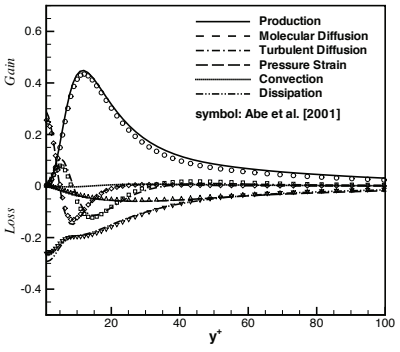


Figure 4. Streamwise turbulence energy budget-Poiseuille flow along the wall bisector

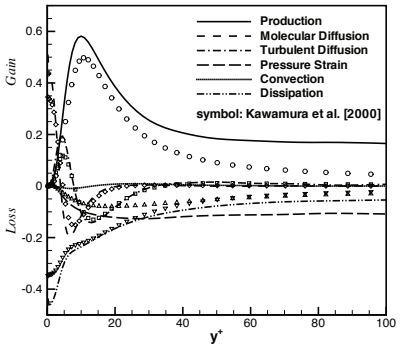


Figure 5. Streamwise turbulence energy budget-Couette flow along the wall bisector

First principles calculations of $N_2 + N^+$ and $N_2 + O$ interaction potentials for the development of a chemical kinetic model

Galina M. Chaban*, Winifred M. Huo, David W. Schwenke ***, Richard L. Jaffe******

*NASA Ames Research Center, Moffett Field, CA 94035, USA
(Tel: 650-604-4995; e-mail: galina.m.chaban@nasa.gov)
**Huo Consulting LLC, 680 Blinn Court, Los Altos, CA, 94024, USA
(e-mail: winifred.m.huo@nasa.gov)
***NASA Ames Research Center, Moffett Field, CA 94035, USA
(e-mail: david.w.schwenke@nasa.gov)
****NASA Ames Research Center, Moffett Field, CA 94035, USA
(e-mail: richard.l.jaffe@nasa.gov)

Abstract: Accurate potential energy surfaces for N_3^+ and N_2O chemical systems are computed using state-of-the-art first principles electronic structure techniques. The computations are performed using parallelized software on the Columbia supercomputer. The data obtained from first principles quantum chemistry calculations provide the basis for the development of more reliable analytical interaction potentials for these chemical systems, which will be used to calculate rate coefficients for dissociation and rotation-vibration energy transfer reactions. Such reactions are important to describe the non-equilibrium flow field under the hypersonics speed conditions of space vehicle reentry. The resulting chemical data will be incorporated into CFD/radiation codes. This should significantly improve chemical kinetics model for the nitrogen/air atmosphere compared to existing empirical models.

Keywords: chemical kinetics database, interaction potentials, first principals calculations, rate coefficients, hypersonics reentry.

1. INTRODUCTION

CFD modeling of hypersonic space flight requires a chemical kinetic database including reaction rate coefficients of such chemical reactions as molecular dissociation and vibration-rotation-translation energy transfer (VRT). These data are needed to capture the complex thermodynamics and nonequilibrium chemistry of real gas behavior and to predict the number densities of multiple species in the flow, their internal energies, and their radiative properties. Our goal is to develop the chemistry database using a first principles approach.

First principles calculation of chemical reaction rates is a multistep process: (1) Determination of the interaction potential among the reactants at a grid of fixed nuclear geometries using highly sophisticated quantum mechanical methods. (2) Determination of a faithful analytical representation of the energies and forces on the grid of nuclear geometries. (3) Determination of cross sections and reaction rate coefficients using quasi-classical trajectories (QCT) calculations based on the analytic interaction potential derived in (2).

Here we present the first principles calculations of the interaction potentials for the $N_2 + N^+$ and $N_2 + O$ systems. These calculations are the first step in the determination of VRT and dissociation rate coefficients for the two systems. The interaction potential of $N_2 + N^+$ will also be used to determine the rate coefficients of the charge exchange reaction $N_2 + N^+ \rightarrow N_2^+ + N$.

2. FIRST PRINCIPLES CALCULATIONS

Accurate potential energy surfaces of N_3^+ and N_2O chemical systems are computed using state-of-the-art multi-configurational electronic structure methodology, such as the multi-configurational self-consistent field (MCSCF) [1], multi-reference configuration interaction (MRCI) and averaged coupled pair functional (ACPF) [2] methods. The computed surfaces include both long-range interactions between the molecules, and the parts of the surface that correspond to dissociation of one of the bonds. The calculations have been carried out at many (>1000) molecular

geometries. Such accurate first principles calculations are very time-consuming and require the use of multi-processor computers in parallel.

An example of one of the potential energy curves for the dissociation of the N-N bond (represented by the bond distance r) in the T-shaped configuration of N_3^+ is shown in Fig. 1. Such calculations have been performed for many different values of distance R between N_2 and N.

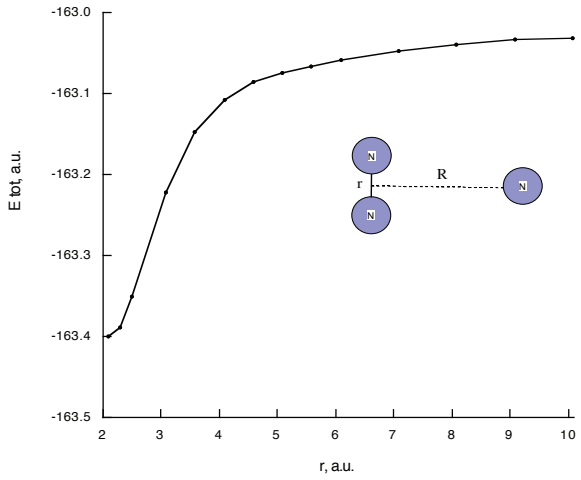


Fig. 1: *Potential energy profile for N-N dissociation of the T-shaped N_3^+ structure with R=4 a.u.*

In the case of N_2O, potential energy surfaces for both singlet and triplet spin states have to be considered when studying the $N_2 + O \Rightarrow N + N + O$ dissociation reaction. As seen from Fig. 2, these electronic states cross in the region of R between 3 a.u. and 4 a.u., and as a result, the triplet states are lower in energy for larger values of R (distance between N_2 and O), while the singlet is the lowest state for short R's. Examples of energy profiles for the singlet state N-N dissociation at a number of distances R are shown in Fig. 3.

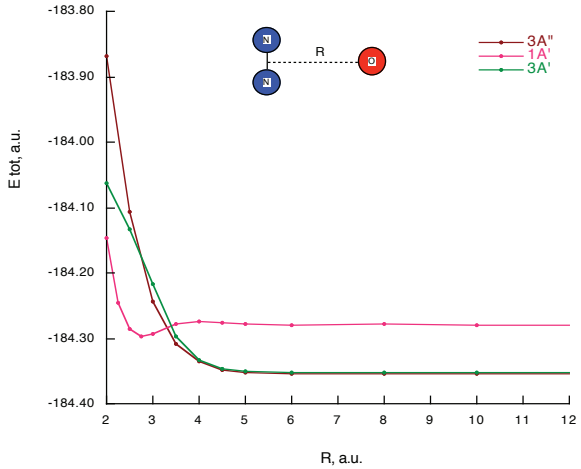


Fig. 2: *Potential energy curves for N_2 + O separation in the T-shaped orientation.*

Many more regions of interaction potentials for the $N_2 + N^+$ and N_2 + O systems have been determined on grids of nuclear geometries. For both systems, the collision dynamics are complicated by the involvement of electronic excited states. Accurate data obtained from first principles quantum chemistry calculations for the N_3^+ and N_2O systems provide the basis for the development of more reliable analytical interaction potentials for these systems. Such interaction potentials will be used to calculate rate coefficients for dissociation and VRT energy transfer. The

data will be incorporated into CFD/radiation codes. This should significantly improve chemical kinetics model for the nitrogen/air atmosphere compared to existing empirical models.

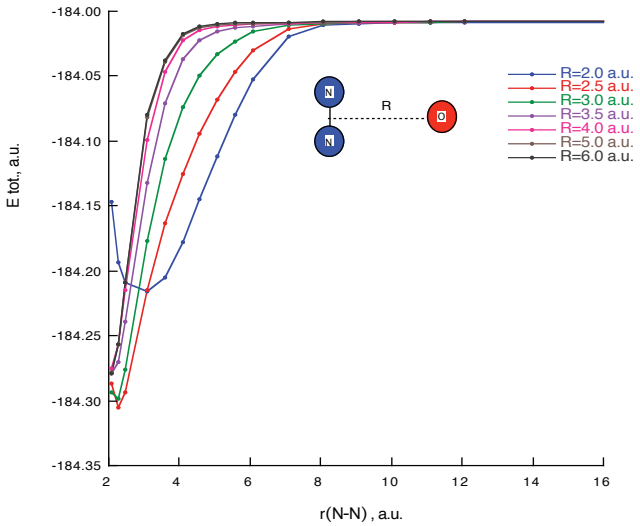


Fig. 3: *Potential energy curves for N-N dissociation of the singlet N_2O.*

3. DIRECT DYNAMICS

In addition to the studies described above, a different approach based on direct "on-the fly" dynamics will be demonstrated. The main advantage of this approach is that the fitting of analytical potential function is avoided and, thus, the multistep first principle computation process (described in the Introduction) becomes a single-step process: potentials and gradients for dynamical trajectories are computed directly using a quantum chemistry program. This approach is not common at present, because it is computationally extremely demanding and relies on the parallel implementation of the code to become feasible. We will demonstrate its future potential by showing examples of the "on-the fly" trajectories obtained using Columbia supercomputer. These calculations have been carried out using the electronic structure package GAMESS (General Atomic and Molecular Electronic Structure System) [3] that has high scalability and very efficient performance (Fig. 4).

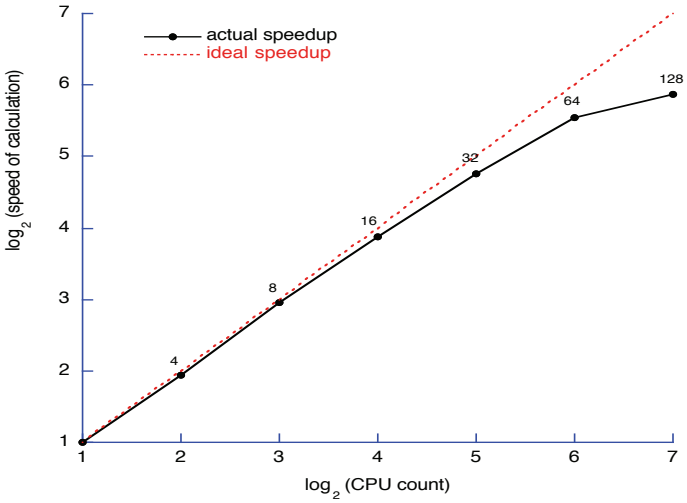


Fig. 4: *Scalability of GAMESS on Columbia supercomputer.*

ACKNOWLEDGMENT

Work presented here is supported by NASA Fundamental Aeronatuics/Hypersonics AAP Project.

REFERENCES

1. Schmidt, M.W. and Gordon, M.S. (1998). The Construction and Interpretation of MCSCF wavefunctions. *Annu. Rev. Phys. Chem.* 49, 233-266.
2. Gdanitz, R. . and Ahlrichs, R. (1988).The averaged coupled-pair functional (ACPF): A size-extensive modification of MRCI(SD), *Chem. Phys. Lett*. 143, 413-420.
3. http://www.msg.chem.iastate.edu/gamess/gamess.html

A GPU-Version Lattice Boltzmann Method for Solving Fluid-Particle Interaction Problems

San-Yih Lin and Yuan-Hung Tai
Department of Aeronautics and Astronautics
National Cheng Kung University
Tainan, Taiwan, ROC
sylin@mail.ncku.edu.tw

ABSTRACT

A GPU-version immersed boundary-lattice Boltzmann method (IB-LBM) is developed on a graphical processing unit (GPU) to simulate fluid-particle interaction problems [1]. This method uses the lattice Boltzmann method to solve the incompressible flow field and the immersed boundary method which applies a direct forcing method to capture the particle motion [2]. By using the parallel device architecture developed by the graphics hardware, an efficiency gain of up to one order of magnitude with respective to the CPU performance of a PC is obtained. A D3Q15 LBM method is used in this paper. A direct forcing method for the IB method is introduced in the LBM method to allow the flow velocity equal to the particle velocity. A flow in a square pipe and a flow over a sphere are simulated to validate the numerical method. As a demonstration of the efficient and capabilities of the new method, sedimentation of two sphere particles in an enclosure is simulated.

Keywords: Immersed boundary method, Lattice Boltzmann method, Fluid-particle interaction, Graphical processing unit, Direct forcing method

NUMERICAL TEST CASES

Three test cases are investigated by the proposed direct-forcing method. They include flow in a square pipe, flow past a fixed sphere and sedimentation of one spherical particle in an enclosure.

1 Flow in a square pipe

To demonstrate the efficiency of the GPU-version IB-LBM method, a flow in a square pipe is simulated. The dimensions of the pipe is L-cm long, W-cm wide and H-cm high. The pipe is simulated using L×W×H lattice points. Fig. 1 shows the velocity and pressure contours. Table 2 shows the computation efficiency compared to the CPU performance.

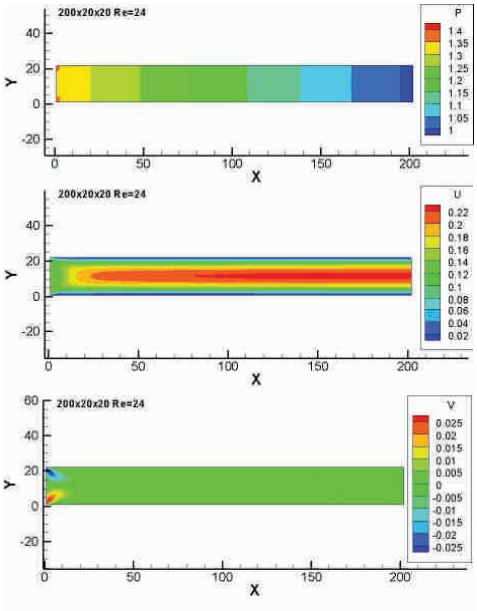


Fig 1. Pressure and velocity contours.

Table 2. computation efficiency compared to the CPU performance.

Case L×W×H	CPU(fortran) (s)	CUDA(s)	ratio
50x20x20	13.72	2.547	5.4
50x50x50	8.22	1.078	7.62
100x100x100	62.0	7.969	7.78
150x100x100	90.01	11.0	8.18

2 Flow past a fixed sphere

Here the flows past a fixed sphere are investigated. It is known that the flow over a sphere is steady and axis symmetry till Re = 210. Four Reynolds numbers Re = 50, 100, 150, and 200 are selected in the test. Fig. 2 shows the computed streamlines with domain, 5×5×5. The sphere is centered at (2.5, 2.5, 2.5) and its diameter is normalized to 1. A uniform Cartesian grid is used with the size $\Delta x = \Delta y = \Delta z = 0.1$. Their vortex structures are very good and attached to sphere nicely. Table 3 show the drag coefficient and vortex length, Lw, for Re = 100, 150, and 200. The compational results are good.

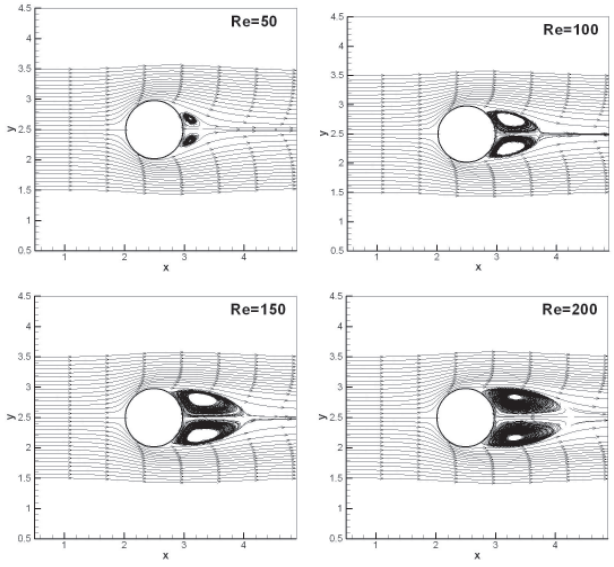


Fig. 2. Computed streamlines for Re = (a) 50, (b) 100, (c) 150, and (d) 200.

Table 3.a The drag coefficient and vortex length for Re = 100.

Re	C_D	Lw
100	1.056	0.891
150	0.877	1.170
200	0.766	1.436

3 Sedimentation of one spherical particle in a wide closure

The sedimentation of a spherical particle in an enclosure is simulated. The dimensions of the enclosure is 10-cm long, 10-cm wide and 16-cm high. The particle commences its motion at a height H = 12 cm from the bottom. The fluid density in the simulations is in the range from 960 to 970 kg/cm^3 and the dynamic viscosity from 0.058 to 0.353 Ns/m^2. The particle has density 1120 kg/m^3 and radius 1.5 cm. The enclosure is simulated using 100×100×160 lattice points and the particle is outlined by 15 lattice points. Figs. 3 and 4 show the particle trajectories and settling velocities between the experimental data and the numerical results for four cases. It observed that the simulation results agree well with the experimental results [3].

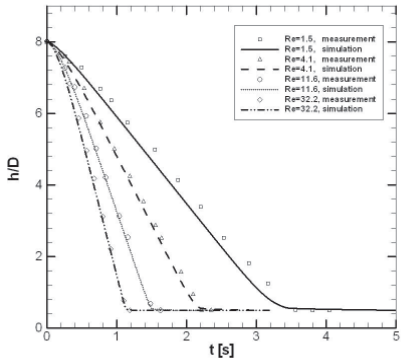


Fig. 3. Comparisons between trajectories by simulations and measurements.

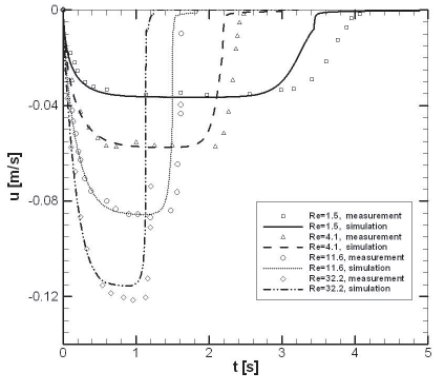


Fig. 4. Comparisons of the measured and simulated settling velocities.

REFERENCES

1. J. Tolke and M. Krafczyk, TeraFOLP Computing on a Desktop PC with GPUs for 3D CFD, International Journal of Computational Fluid Dynamics, Vol. 22. No. 7, pp. 443-456, 2008.

2. C. S. Peskin, Numerical Analysis of Blood Flow in the Heart, Journal of Computational Physics, Vol. 25, pp. 220-252, 1977.

3. A. ten Cate, C. H. Nieuestad, J. J. Derksen, and H. E. A. Van den Akker, Particle Imaging Velocimetry Experiments and Lattice-Boltzmann Simulations on a Single Settling under Gravity, Physical Fluids, Vol. 14, pp. 4012-4025, 2002.

Lattice Boltzmann simulation of particle suspensions under the influence of dynamic force fields

E. Monaco*, G. Brenner*

**Institute of Applied Mechanics, Adolph-Roemer Str. 2A, Clausthal University, 38678 Clausthal-Zellerfeld, Germany*
(Tel: +49(0)5323722515 ; e-mail: ernesto.monaco@tu-clausthal.de)

Abstract: The goal of the present work is to quantify the influence of shear or acceleration forces on the sedimentation of particles. The transport of suspended particles in a liquid medium is studied by means of the lattice-Boltzmann Method for the continuous phase in combination with a molecular-dynamics approach to describe the motion of suspended particles. Therefore, interactions among particles and between fluid and particles are modeled at particle scale. Phenomena at bulk scale are obtained by considering a sufficiently large representative elementary volume.

Keywords: Particle suspensions, lattice Boltzmann method, sedimentation control.

1. INTRODUCTION

Solid particles like mineral, polymer or drugs are often transported as suspensions in a liquid medium. The transport operations should preserve the stability of the suspension, therefore the control of the sedimentation induced by gravity is a fundamental problem to address. As pointed out in [1], the stability of a suspension under dynamic forces is different from that under static conditions. Different kinds of dynamic forces can occur and have been studied in the literature [1-2]. Numerical simulations may greatly improve the understanding of these phenomena, since they allow a very detailed assessment of the state of the suspension. In that context, in [3], the stability of a dilute suspension (particle concentration up to 8 wt.%) of spherical particles under horizontal movement of the system was studied by means of numerical simulations in addition to experiments. The goal of this study is to demonstrate the ability of the lattice-Boltzmann Method (LBM) to describe the dynamics of a suspension under time dependant external forces.

2. COMPUTATIONAL APPROACH

In order to perform numerical simulations of particle suspension, the following requirements have to be matched by the computational approach: (i) the hydrodynamic interactions at solid-fluid interfaces have to be reproduced with reasonable accuracy and (ii) the motion and interaction of thousands of particles have to be computed at reasonable computational costs. In this work the LBM [4] is employed for the modeling of the continuous fluid phase. The LBM derives from a description of the fluid state based on the kinetic theory. In essence, the LBM is a discretization of the Boltzmann equation in terms of discrete velocity distribution functions of particle populations. The algorithm basically consists of a streaming step involving next neighbour nodes and a purely local collision steps, making the method particularly suited for massively parallel computations. The advantage of LBM as compared to other numerical schemes is its ability to simulate dynamic processes at fairly low computational costs. The molecular dynamics approach is used to update particle positions. It consists of Lagrangian tracking of individual spherical particles and considers the conservation of linear and angular momentum as well as the momentum exchange due to particle collisions. The coupling between fluid and particle motion is achieved by applying the immersed boundary approach on a Cartesian mesh. The most common way of modeling the no-slip boundary condition is the so-called "Bounce-Back" scheme [4], in which distribution functions are reflected back into the computational domain after the streaming step. This scheme was generalized by Ladd [5, 6] for moving boundaries as immersed particles. This formulation allows a simple calculation of forces and torques acting on every particle, except when two or more particles are in contact: in this case, the hydrodynamic (lubrification) forces have to be modeled and applied to the scheme, following the approach proposed in [7].

3. RESULTS

3.1 Validation of the code

For validating of the LBM code a comparison is made between experiments reported in [8]. The motion of a single particle sedimenting in a viscous fluid is considered. In [8] PIV (particle image velocimetry) is used to measure the fluid velocity around the particle. Besides that the trajectories of the particles as well as sedimentation velocities are obtained for various Reynolds numbers in the range of 1.5 and 32. Here, the Reynolds number is defined in terms of the terminal velocity and the diameter of the particle. Two cases for Re=1.5 and Re=11.6 are shown. In Figure 1 the trajectories of the particle for these cases are presented as well as the particle velocities. The sequence in Figure 2 and Figure 3 show respectively the contours of vertical velocity in different phases of the sedimentation process and a detail of the velocity vectors around the particle when the bottom has almost been reached. In the present both calculations the spherical particle is resolved by 4 lattice sites while the overall size of the domain is 55x87x55 nodes. For low Reynolds numbers after about one second a constant terminal velocity is obtained. As the particle is approaching the wall due to lubrification effects the velocity is slowly decreasing. This effect is well resolved using the LBM code. For higher Reynolds numbers the terminal velocity is significantly higher but the damping of the motion close to the wall is less distinctive. The full paper will also report on the problem of swapping trajectories if more than one particle is present as well as in the influence of a mean shear in the flow [9].

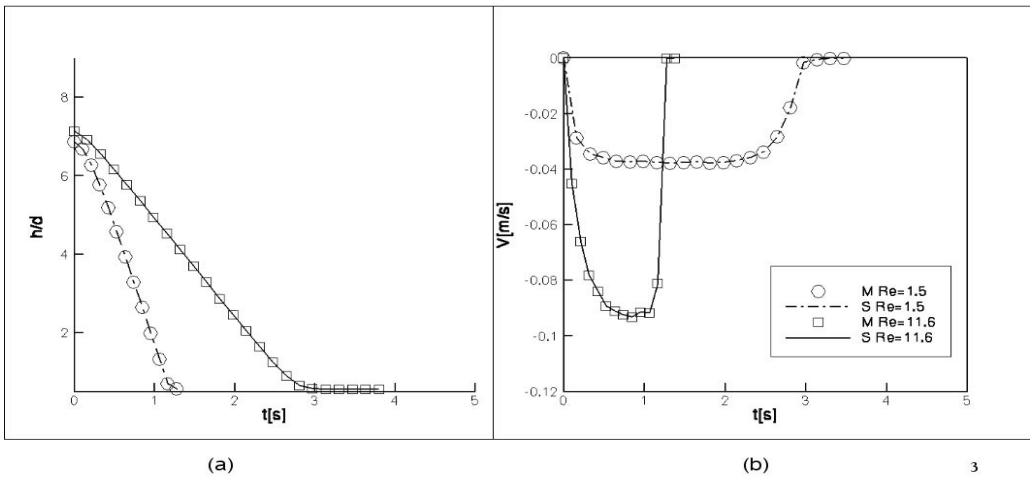


Fig. 1: Simulations (S) versus experimental measures (M) for trajectories (a) and sedimentation velocities (b). Re=1.5 and Re=11.6 cases are considered.

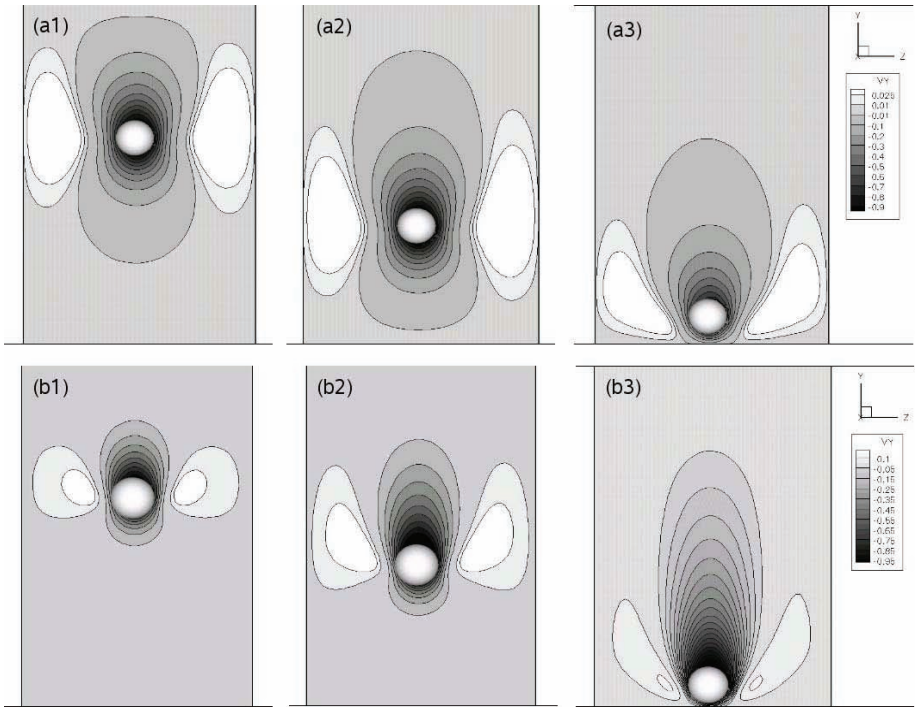


Fig. 2: Comparison of the flow field of sedimenting sphere at a) Re=1.5 and B) Re=11.6

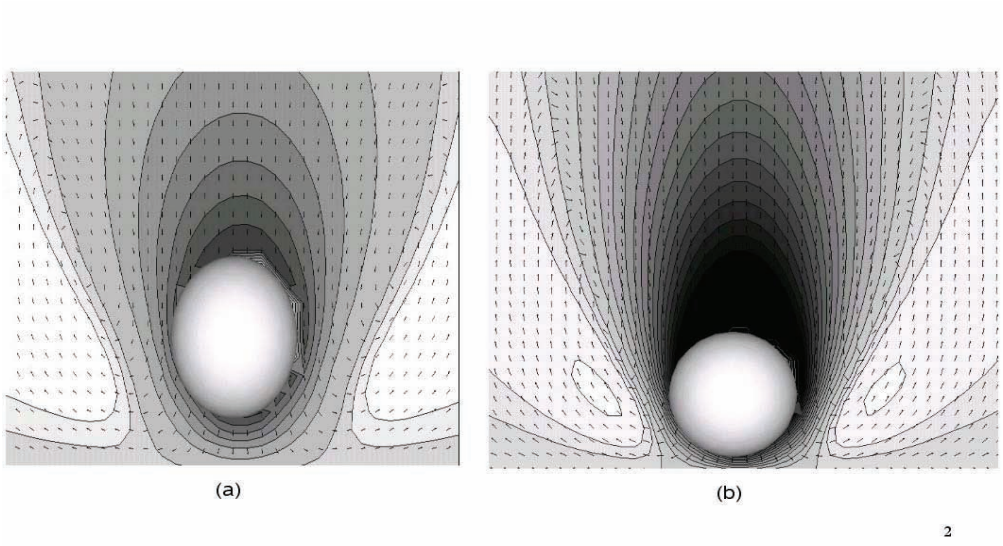


Fig. 3: Comparison of the velocity field of sedimenting sphere at a) Re=1.5 and B) Re=11.6 while approaching the bottom wall

3.2 Particle suspensions under the influence of dynamic force fields

The focus of the paper is on evaluating the influence of amplitude and frequency of a periodic forcing on the settling of a large cluster of spherical monodispersed particles due to gravity. The problem is characterized by means of non-dimensional parameters in order to determine the influence of fluid/particle density ratio or particle size. The challenge of the present problem is the disparity of scales. On the one side, the computational mesh has to capture the details in the flow around a single particle which is discretized with at least 4-5 lattice cells. On the other hand, the effect of formation of secondary flows is only observed for reasonable large container sizes. Besides that, a

sufficiently large number of particles have to be considered in order to obtain reasonable statistics.

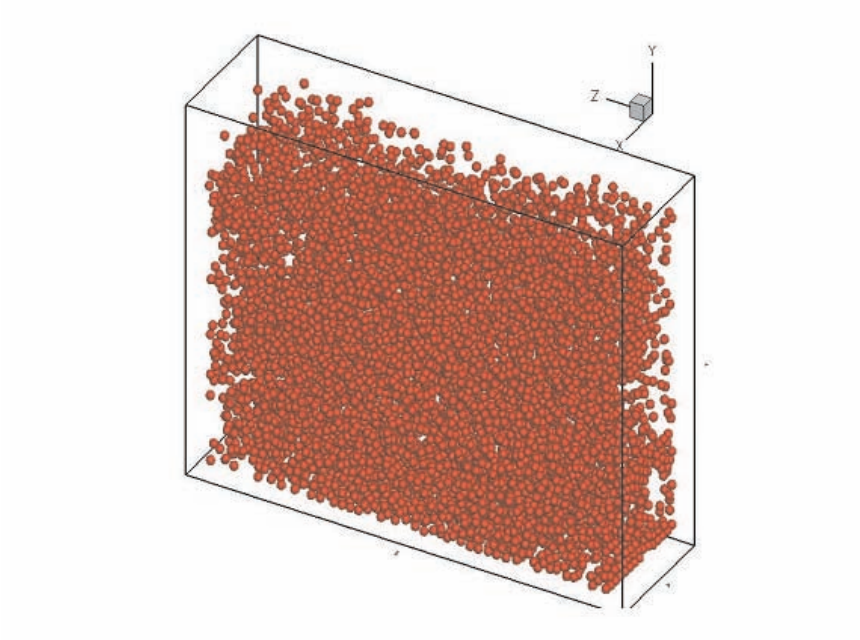


Fig. 4: Final particles configuration of a suspension of 7860 particles. The oscillation period is 320 and the amplitude is 0.05 (in LBM units).

In Figure 4, the particle positions are represented for a configuration of 7860 particles, each of them resolved with a diameter of 5 in lattice units. Figure 5 presents the instantaneous velocity field of the particles. In this case the particle/fluid density ratio is 1.5. The circulation flow induced by the periodical movement of walls is evident. Good comparisons with experimental results reported in [3] are achieved. Keeping the oscillation amplitude constant, the application of the horizontal movement clearly counteracts the sedimentation induced by gravity if the oscillation period is below a critical value. Thus, the stabilizing action is more effective as the oscillation frequency is increased. The full paper will in particular discuss the influence on oscillation amplitude and fluid/particle density ratio and the spatial resolution on the sedimentation control.

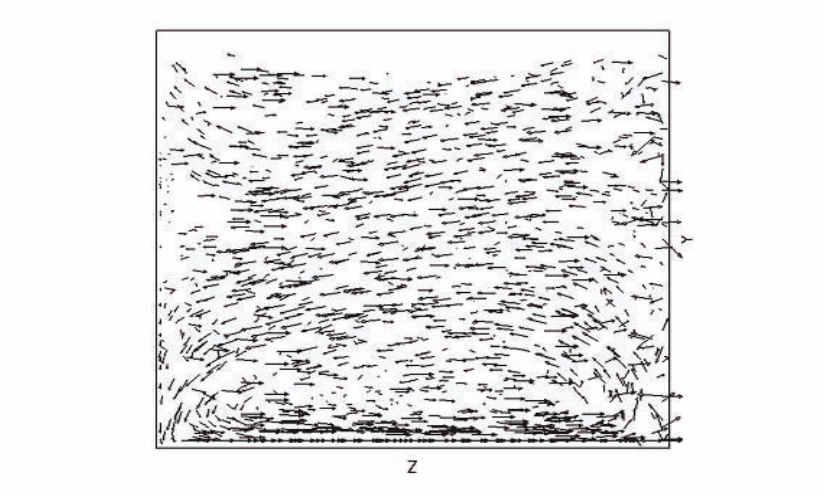


Fig. 5: Instantaneous velocity distribution of a suspension of 7860 particles. The oscillation period is 320 and the amplitude is 0.05 (in LBM units).

REFERENCES

1. Matsuno, Y., Akagi, S. and Kasai, H. (1991). Fluid motion and settling of particles of pulverized coal in the tank of a COM carrier ship. Liquid Solid Flows, FED volume 118, pp. 255–262, ASME.
2. Usui, H., Saeki, T. and Mori, T. (1995) Storage transportation of coal water mixtures, 5th International Conference on Bulk Materials Storage, Handling and Transportation, pp. 39–46.
3. Yoshida, H., Nurtono, T. and Kunihiro, F. (2005). A new method for the control of dilute suspension sedimentation by horizontal movement. Powder Technology, volume 150 (1), pp.9-19, Elsevier.
4. Chen, S. and Doolen, G. (1999). .Annual Review of Fluid Mechanics, volume 31.
5. Ladd, A. J. C. (1994). Numerical Simulations of Particulate Suspensions via a discretized Boltzmann Equation. Part I. Theoretical Foundation. J Journal of Fluid Mechanics, volume 271. Cambridge University Press.
6. Ladd, A. J. C. (1994). Numerical Simulations of Particulate Suspensions via a discretized Boltzmann Equation. Part II. Numerical results. Journal of Fluid Mechanics, volume 271. Cambridge University Press.
7. Nguyen, N. Q. and Ladd, A. J. C. (2002). Lubrication corrections for lattice-Boltzmann simulations of particle suspensions . Physical Review E, volume 66. American Physical Society.
8. ten Cate, A., Nieuwstad, C. H., Derksen, C. C., Van den Akker, H. E. A. (2002). Particle imaging velocimetry experiments and lattice-Boltzmann simulations on a single sphere settling under gravity. Physics of Fluids, volume 14(11), pp 4012-4025.
9. Zurita-Gotor, M., Blawzdiewicz, J. and Wajnryb, E. (2007). Swapping trajectories: a new wall-induced cross-streamline particle migration mechanism in a dilute suspension of spheres. Journal of Fluid Mechanics, volume 592. Cambridge University Press.

OpenMP Parallelization of Linear Fixed-Point Methods Applied to the Pressure Poisson Equation

J. M. McDonough*, J. B. Polly, and J. P. Strodtbeck****

**Departments of Mechanical Engineering and Mathematics, University of Kentucky, Lexington, KY, 40506-0503, USA*
Tel: 859-257-6336, x80657; e-mail: jmmcd@uky.edu

***Department of Mechanical Engineering, University of Kentucky, Lexington, KY, 40506-0503, USA*
Tel: 859-257-6336, x80670; e-mail: jamespolly@gmail.com

***Department of Mechanical Engineering, University of Kentucky, Lexington, KY, 40506-0503, USA*
Tel: 859-257-6336, x80670; e-mail: jpstro2@uky.edu

Abstract: We present results from very simple OpenMP parallelization of a 3-D SOR algorithm applied to a pressure Poisson equation on a shared-memory IBM P-Series machine. It is observed that four processors is the maximum with which improvements in speed up can be achieved.

Keywords: Poisson equations, sparse matrix solvers, successive overrelaxation.

1. INTRODUCTION

It is common knowledge that arithmetic required for CFD of incompressible flow problems is dominated by that used for solution of the pressure Poisson equation and, hence, satisfaction of the divergence-free constraint—*i.e.*, mass conservation. There have been many classes of methods applied to this problem, including the following: basic relaxation, Krylov-subspace-based methods of various types (*e.g.*, GMRES and BiCGStab(ℓ)), multi-grid and domain decomposition methods, and various combinations of these. None of these has proven to be entirely satisfactory over wide ranges of elliptic problems, and there still is considerable controversy regarding which type of method to use for any particular problem. It is, in fact, likely that no single method will be superior for all pressure Poisson problems.

But there are general characteristics of these classes of methods that can be considered when making comparisons amongst them, and between individual techniques, either within the same class, or across classes. These are:

i) required total arithmetic ⇒ run time in serial mode;

ii) ease of implementation and use ⇒ less human time expenditure;

iii) parallelizability, including parallel efficiencies ⇒ faster turn around;

iv) effects of problem size on all of the preceding.

The first of these consists of the product of number of required iterations and arithmetic per iteration. The second involves algorithm intricacies (including code for parallel constructs) in the case of implementation, and selection of iteration parameters in the event that the entire solution algorithm is being applied to an actual CFD (or other) problem by an "end user." Parallelizability is, of course, algorithm dependent. But it is also machine dependent and individual problem size dependent; finally, parallel speed ups are significantly influenced by choice of OpenMP or MPI in the context of computing hardware employed, *i.e.*, clusters or SMPs.

Effects of problem size arise in essentially all of the above problem aspects, and they are most noticeable (as problems become very large) in asymptotic convergence rates of iteration procedures, in cache utilization

of any particular computer, and in specific details of parallelization efforts. "Large" problems generally result in poor asymptotic convergence rates and degraded cache utilization, but improved parallel effectiveness. Thus, within the confines of a given processor and a choice between OpenMP and MPI, our goal must be to mitigate both asymptotic convergence rate and cache-miss problems while achieving good parallel speed ups. In this study we investigate use of OpenMP applied to sparse linear systems arising from second-order centered discretizations of the 3-D Poisson equation in the context of various forms of successive overrelaxation. The study will be conducted on at least two different machines: one consisting of clusters and the other featuring shared memory, and it will entail comparisons of different domain decomposition strategies. This abstract contains only preliminary results from this ongoing study; the remainder will be presented at the *Parallel CFD Conference*.

2. ANALYSIS

The pressure Poisson equation can be expressed as

$$\Delta p = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega \subseteq \mathbb{R}^3\,, \tag{1}$$

with p being pressure and f corresponding to either an appropriately scaled divergence of the velocity field or divergence of the advective terms of the Navier–Stokes (N.–S.) equations, depending, respectively, upon whether one is constructing a projector to satisfy the divergence-free condition or computing the true physical pressure which appears in the N.–S. equations (see, *e.g.*, [1], for details). In (1) $\boldsymbol{x} \equiv (x, y, z)^T$ is the Cartesian coordinate vector so that for purposes herein the Laplacian is simply

$$\Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\,. \tag{2}$$

In the present study we will take Ω to be a right-angled parallelepiped. Finally, we note that for well posedness, boundary conditions must be provided at all points of $\partial\Omega$, the boundary of Ω. For the problem considered in this abstract these will all be of Dirichlet type, but the final paper will contain results from mixes of Dirichlet and Neumann conditions typically found in CFD problems.

We employ standard second-order centered-difference approximations to the differential operators of (1) and (2) on a uniform grid of $N = N_x \cdot N_y \cdot N_z$ points with $h_x = (b_x - a_x)/(N_x - 1)$, *etc.*, in which $b_x - a_x$ is the length of the segment of $\partial\Omega$ in the x direction. For a Dirichlet problem such as being considered here, this leads to a system of discrete equations of the form

$$\frac{p_{i-1,j,k} - 2p_{i,j,k} + p_{i+1,j,k}}{h_x^2} + \frac{p_{i,j-1,k} - 2p_{i,j,k} + p_{i,j+1,k}}{h_y^2} + \frac{p_{i,j,k-1} - 2p_{i,j,k} + p_{i,j,k+1}}{h_z^2} = f_{i,j,k}\,, \tag{3}$$

$\forall\ i, j, k = 2, 3, \ldots, N_x - 1, N_y - 1, N_z - 1$, with boundary values prescribed (and therefore, formally, not calculated) on $\partial\Omega$.

For problems such as those embodied in (3), it is extremely simple (not much human time required) to construct a successive overrelaxation (SOR) procedure formally corresponding to a linear fixed-point iteration of the form

$$\boldsymbol{p}^{(n+1)} = \boldsymbol{\mathcal{L}}_\omega \boldsymbol{p}^{(n)} + \boldsymbol{k}_\omega\,, \tag{4}$$

where $\boldsymbol{p} = (p_{1,1,1}, \ldots, p_{i,j,k}, \ldots, p_{N_x,N_y,N_z})^T$ is the solution vector of discrete pressure values, and $\boldsymbol{\mathcal{L}}_\omega$ is the SOR iteration matrix (see [2] for an encyclopedic treatment). It is clear that once $\boldsymbol{\mathcal{L}}_\omega$ has been constructed (which is not actually done in a computational algorithm) only one matrix multiplication is required per iteration, implying $\sim\mathcal{O}(N)$ arithmetic operations per iteration with a fairly small constant associated with $\mathcal{O}$. Moreover, it is known (see [2]) that the total number of iterations needed to achieve a specified level of convergence scales as $\mathcal{O}(N^{1/3})$ provided the iteration parameter, ω, is the unique optimal one, ω_b. Hence, in this case total arithmetic is $\sim\ \mathcal{O}(N^{1.333\ldots})$. Furthermore, we note that for Laplace-Poisson/Dirichlet problems, as considered in this abstract, ω_b is known exactly and depends only on problem geometry and discretization. We remark that although multigrid and various Krylov-subspace-based methods are often

viewed as being more efficient (and the latter does not require specification of any parameters), they are much more difficult to implement, and, particularly in the case of the latter of these, their asymptotic convergence rates deteriorate significantly in applications to large problems of present-day interest (see [3]). Thus, we believe, within the context of points *i*) and *ii*) of the preceding section, that the approach employed here, basic linear fixed-point iteration, is worth considering—along with, of course, parallelization.

3. PARALLELIZATION

There are at least two distinct approaches to parallelization of relaxation methods in 3D, with various options within each of these. Figure 1 provides a schematic of these in the context of the problem considered for this abstract (solution of (3) on rectangular domains). Both parts of this figure represent forms of domain decomposition strategies; but they are quite different. In part (a) of the figure we display an approach that relies on 3-D subdomains while the method

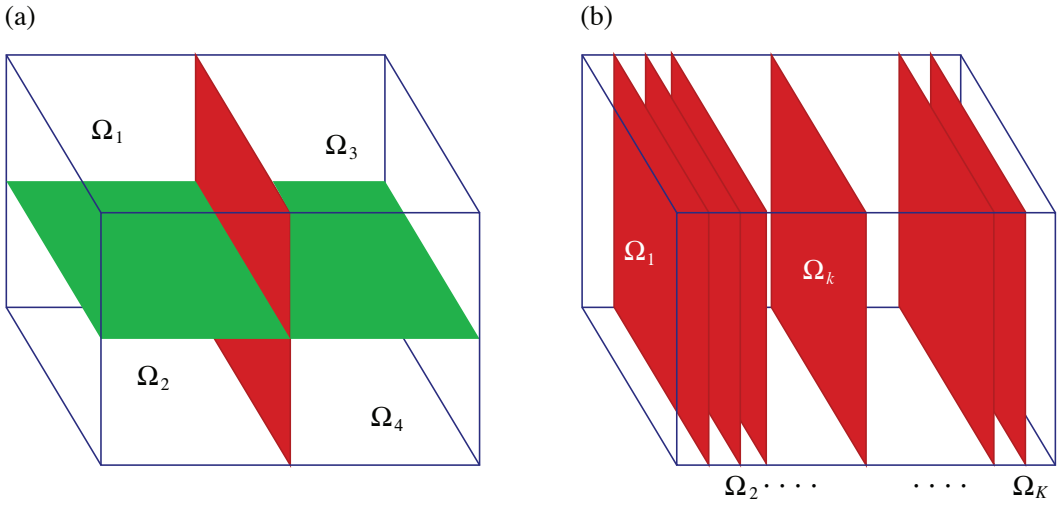


FIG. 1: *Parallelization strategies for relaxation-like algorithms on regular domains.*

shown in part (b) utilizes 2-D subdomains. We remark that the first of these will show some advantage for SOR-related methods because their convergence rates are higher in 3-D. Moreover, this approach lends itself to methods (extended to 3D) analogous to the 2-D techniques involving approximate Schur complements studied in [4]. On the other hand, use of 2-D subdomains as depicted in part (b) is easily and naturally coded in the context of OpenMP, as initially done herein. Furthermore, there can be some further advantages to treatment of non-Dirichlet boundary conditions with this approach. Beyond this, we observe that the 3-D formulation will typically utilize fewer processors, which is possibly an advantage when using OpenMP because this technique generally does not scale well beyond about 16 processors. We emphasize that in the current abstract, only the approach implied by part (b) of the figure has been preliminarily investigated. Further results from both types of methods will be presented at the conference and in the final paper.

4. RESULTS

Results presented here deal only with parallel speed ups. In particular, the main problem being solved is a trivial one consisting of $f \equiv 0$ in $\Omega \backslash \partial\Omega$ in (1), and $p \equiv 1$ on $\partial\Omega$. It is easily seen that $p \equiv 1$ is the unique exact solution. Thus, there is no truncation error associated with the discretization (3), permitting precise assessment of iteration and rounding errors in the current parallel environment. At the same time, since convergence rates, in general, depend only on the spectral radius of the iteration matrix, it is clear that for any constant-coefficient problem with smooth $f(\boldsymbol{x})$ and constant boundary values the method will behave in the same way as indicated for the present simple problem.

Calculations were performed on the unit cube, $\Omega \equiv [0,1]\times[0,1]\times[0,1]$, and iterates were required to satisfy a convergence tolerance $\epsilon = 10^{-8}$ applied to the max norm of the iteration error $d_{ijk}^{(n)} \equiv p_{ijk}^{(n+1)} - p_{ijk}^{(n)}$, where n is the iteration counter. Discretizations of this domain consisted of 101^3, 201^3 and 301^3 points. Figure 2 displays a convergence history for the 201^3-point grid for optimal SOR ($\omega_b = 1.969071$) and the widely-used $\omega = 1$ corresponding to Gauss–Seidel iterations. It is clear from this figure that one should essentially never employ $\omega = 1$ for typical Poisson equation solutions, as emphasized in [3]. Indeed, this figure contains only a small portion of the Gauss–Seidel convergence history (in order to somewhat more clearly display the optimal SOR result). Iterations converged in 662 iterations (starting from $p^{(0)} \equiv 0$) for optimal SOR and required 43925 iterations for Gauss–Seidel (from the same initial guess).

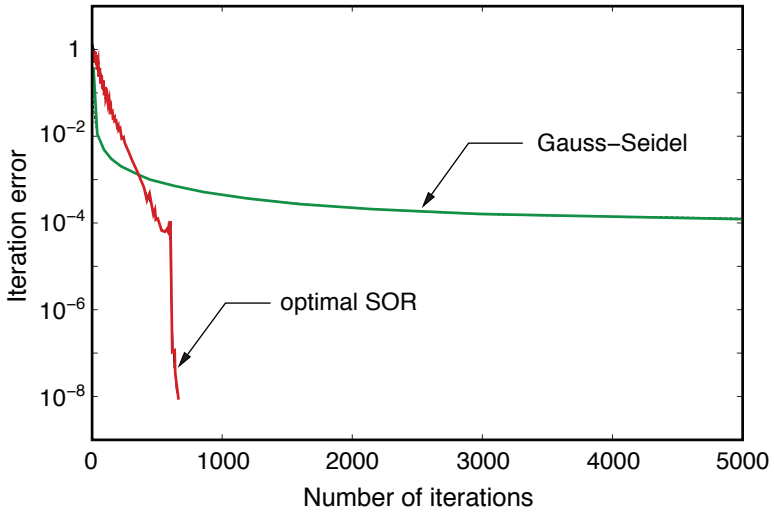


FIG. 2: *Convergence history of relaxation methods.*

Finally, Fig. 3 presents the main results from this preliminary study, *viz.*, parallel speed ups using OpenMP in a simple parallelization of the form implied by Fig. 1 (b). These calculations were performed on an IBM P-Series shared-memory machine at the *University of Kentucky Computing Center*. Overall, this computer consists of 128 IBM Power 5 processors and 1 TB of shared memory. We emphasize that the results displayed here have been produced with no attempted optimization of OpenMP constructs—less than five minutes of human time were expended in setting up parallelization of the two inner loops (corresponding to the planes depicted in Fig. 1 (b)) of the three-loop evaluation of the SOR iteration formula.

It can be seen from Fig. 3 that one would not likely gain from using more than four processors for any of the three problem sizes employed here. In fact, run time (wall clock) with four processors was actually slightly longer than with two in the 101^3 grid-point case, which showed slight super speed up, probably due to improved cache utilization. The 201^3-point calculation shows perfect speed up between one and two processors, but little further speed up when four processors are used. Finally, for the 301^3 grid point problem, speed up between one and two processors is significantly inferior to that of the other two smaller problems, again probably reflecting cache utilization—but this time the problem is sufficiently large that splitting it between only two processors does not improve cache efficiency as it did for the smaller problems. For this fairly large problem, employing four processors showed at least an observable, though not very significant, speed up over that of two processors.

5. CONCLUSIONS/FUTURE WORK

The results of this preliminary study show that linear fixed-point iteration procedures in the form of optimal SOR can be readily applied to obtain efficient solutions to 3-D Laplace/Dirichlet problems analogous to the pressure Poisson equation of incompressible CFD in that they can be easily parallelized via OpenMP; but

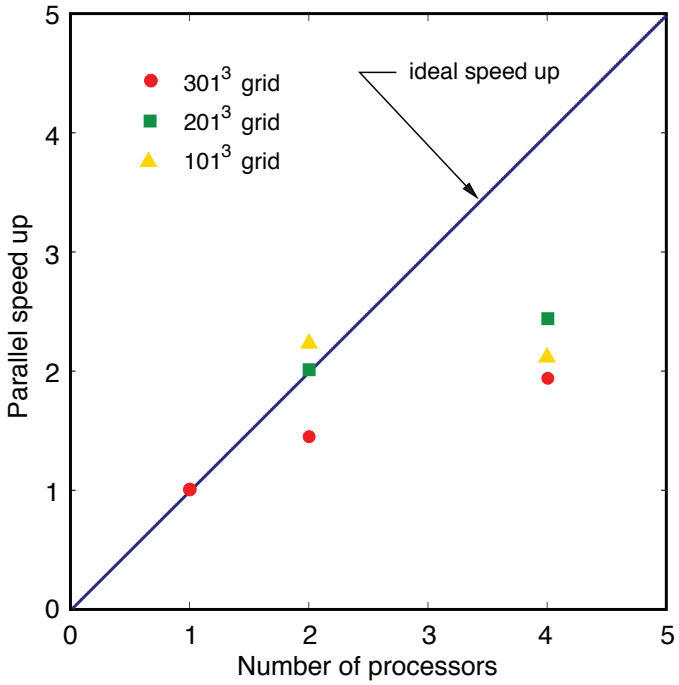


FIG. 3: *Parallel speed ups for relaxation-like algorithms on rectangular domains.*

speed ups are not especially good when more than two processors are utilized. Future work, to be reported at the conference and in the formal proceedings, will involve more sophisticated OpenMP parallelizations, more realistic pressure Poisson equation problems (*e.g.*, pressure in a 3-D duct with inflow and outflow), and assessment of OpenMP performance on an IBM cluster of Intel processors.

Acknowledgements: JPS expresses his thanks to the *Kentucky Space Grant Consortium* (associated with NASA) for a graduate fellowship that has in part provided support for his research.

REFERENCES

[1] C. Foias, O. Manley, R. Rosa and R. Temam (2001), *Navier–Stokes Equations and Turbulence*, Cambridge University Press.

[2] D. M. Young (1971), *Iterative Solution of Large Linear Systems*, Academic Press, New York.

[3] J. M. McDonough, J. B. Polly and J. P. Strodtbeck (2009), Assessment of Iterative Methods for Large Linear Systems, in progress.

[4] A. W. Schueller and J. M. McDonough (2002), A multilevel, parallel, domain decomposition, finite-difference Poisson solver, in *PARALLEL COMPUTATIONAL FLUID DYNAMICS, Practice and Theory*, P. Wilders *et al.* (Eds.), Elsevier, Amsterdam, pp 315–322.

PARALLEL AND ADAPTIVE FINITE ELEMENT CFD SIMULATIONS

TROND KVAMSDAL*, KNUT M. OKSTAD*, RUNAR HOLDAHL* AND BJARTE HÆGLAND†

*SINTEF Information and Communication Technology
N-7465 Trondheim, Norway
Email: Trond.Kvamsdal@sintef.no, Knut.Morten.Okstad@sintef.no, Runar.Holdahl@sintef.no

†SINTEF Fisheries and Aquaculture
N-7465 Trondheim, Norway
Email: bjarteha@gmail.com

Key words: CFD simulations, Parallel grid refinement, Adaptive methods.

Summary. *The implementation of an adaptive grid refinement procedure in a parallel CFD simulation environment is discussed. The fluid solver,* Vista-CFD*, is based on the software toolbox* Diffpack*. The adaptive procedure is demonstrated on a 2D benchmark case, the Lid-driven cavity flow.*

1 INTRODUCTION

In many engineering applications, the task of assessing hydrodynamic loading quantities on structures in or under water is a key challenge. This can be addressed either by performing expensive physical tests, or through numerical simulation using Computational Fluid Dynamics (CFD) based on the Finite Element Method (FEM) or similar discretization approaches. However, numerical simulations—in terms of computer power—can be expensive too, for sufficient accuracy and reliability on the results to be obtained.

Efficient simulations of 3D fluid flow around a submerged structure is only attainable when distributing the computations on a parallel processing environment. Then we might be able to add a sufficient amount of degrees of freedoms (DOFs) in order to resolve boundary layers and similar small features that typically appear in the solution, with satisfactory accuracy. Even higher accuracy and/or efficiency could be obtained if we were able to distribute the huge amount of DOFs in an optimal way such that more DOFs are devoted to the regions where the flow is the most complex, and less to the more calm regions. Adaptive grid refinement based on *a posteriori* error estimates can be used to obtain such an optimal distribution of the DOFs, more or less automatically.

The combination of parallel simulation on a distributed grid with an adaptive modification of the grid based on the results is however a challenge by itself. One is then faced with added complexities such as dynamic load re-balancing among the computing nodes, solution transfer between different grids, as well as model data migration between the

computer nodes, all at certain time steps within the time integration procedure.

The current study is a step towards such parallel adaptive CFD analyses. We base our investigation on the in-house CFD-simulation package, *Vista-CFD* [1] which has been developed at SINTEF and NTNU over the last decade through several PhD-projects. Vista-CFD is again based on the object-oriented toolbox *Diffpack* [2, 3]. Herein, we discuss some issues regarding the grid-refinement itself and the subsequent re-balancing of the distributed grids which takes place before the simulation process can resume. We demonstrate the ideas by simulation a simple 2D benchmark case, the well-known cavity flow problem over a square domain.

2 THE VISTA CFD SIMULATION ENVIRONMENT

Vista-CFD is an object-oriented toolbox for coupled problems where solving the incompressible Navier–Stokes equations is a major part. By toolbox, we here mean that on one hand it is a CFD-solver for a given set of problem types, but it is also a development platform, a kind of a library of CFD modules or predefined software building blocks, to build simulators for specific problems in a flexible and efficient way by the user directly.

Vista-CFD is a fully parallelized FE code where the incompressible Navier–Stokes equation is solved using either mixed elements or equal order elements using operator splitting (continuous projection method). Turbulence may be accounted for through the Spalart-Allmaras model, which is implemented as a separate sub-problem solver that interacts with the Navier–Stokes solver in a classical staggering solution procedure.

2.1 A posteriori error estimation for Navier–Stokes simulations

The basis for the adaptive grid-refinement performed in this work is a set of error indicators computed for each element based on an error estimate. Recalling that the incompressible Navier–Stokes equations may be written

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} + \nabla p - \frac{1}{Re}(\Delta \boldsymbol{u}) = \boldsymbol{0} \quad (1)$$

$$\nabla \cdot \boldsymbol{u} = \boldsymbol{0} \quad (2)$$

where $\boldsymbol{u}$ and p are the the velocity and pressure for the fluid, respectively and Re is the Reynolds number, we propose estimating the error of the associated finite element solution $(\boldsymbol{u}^h, p^h)$ through

$$\|\boldsymbol{e}^*\|_\Omega = \|\boldsymbol{u}^* - \boldsymbol{u}^h\|_{E(\Omega)} + \|\nabla\cdot\boldsymbol{u}^h\|_{L_2(\Omega)} \quad (3)$$

where $\boldsymbol{u}^*$ represents an improved version of $\boldsymbol{u}^h$. The first term on the right hand side of Equation (3) is the energy norm of the Navier–Stokes equations, while the second term measures the error in the divergence of the FE velocity field $\boldsymbol{u}^h$ and is thus related to the error in the pressure, p^h.

The improved field is typically established by smoothing the gradient field, $\nabla \boldsymbol{u}^h$. This can be done through Zienkiewicz–Zhu-style patch recovery methods. However, in the current study we rely on simple nodal averaging as the smoothing procedure.

3 ADAPTIVE REFINEMENT OF A DISTRIBUTED FE GRID

The basic principle for adaptive grid refinement is to obtain a computational grid with equidistribution of the estimated error. Such a grid is optimal as it provides results of a given accuracy with a minimum number of DOFs. We have chosen to refine a prescribed portion of the elements, i.e., $\beta \cdot N$ having the greatest element relative error.

The Diffpack toolbox is already equipped with methods for adaptive refinement of simplex elements (triangles in 2D and tetrahedrons in 3D), for serial (non-parallel) applications. In this process, an element is refined by first adding a new node at the mid-point of each of its three (or six in 3D) edges, and then replacing the old element by a set of new elements connecting the existing and the new nodes. When refining one element in this manner, one must also pay attention to the neighboring element(s) sharing the edges that are subdivided, in order to avoid grids with 'hanging nodes'.

However, since the grid now is distributed over a several processors, we are faced with two additional challenges:

1. To maintain matching grids across neighboring sub-grids when refining the (possibly overlapping) sub-grids locally on each processor.

2. Redistribution of the local grids after grid refinement, in order to maintain optimal load balancing among the processors when continuing the flow simulation.

The first challenge is addressed by establishing a global list of element edges (identified by their two global node numbers), containing all edges that are present on more than one processor (in the following, we denote these edges as *overlap edges*. When an element is marked for refinement, and one of its edges are among the overlap edges, a message is sent to the corresponding neighbor processor telling it to refine that particular edge too. This way, we ensure that all overlap edges are refined similarly on all processors. When the edge-refinement is finished, we can continue creating the new, refined, elements using the existing method for serial applications.

In order to re-balance the locally refined grids, we employ the *ParMETIS* library [4] to compute a new distribution of the elements in the refined grid. ParMETIS takes as input the adjacency graph of the local grid on each processor and outputs a list of processor IDs for each element telling which processor each element rather should be on. Based on this list, we communicate nodal coordinates and associated result field quantities among the processors, such that each one is able to reestablish a local sub-grid and result fields, according to the computed partitioning.

It should be noted that this process is somewhat more complicated than the construction of the initial sub-grids, since in the latter process we typically have a global grid

established as a starting point, and now we also have to deal with result fields in addition to the grid itself.

4 NUMERICAL RESULTS

The parallel adaptive procedure has been tested on a simple 2D case; the well-known Cavity flow problem. It consists of a square domain, with wall conditions on three boundaries and a prescribed constant shear flow on the fourth (the upper horizontal boundary).

We have run this problem adaptively on 16 processors. The initial global grid contains $40 \times 40 \times 2 = 3200$ triangular elements. The simulation was performed using a time step size if $\mathrm{d}t = 0.005$ [s] and doing grid refinements in an interval of 2.0 [s]. The initial grid and the first five refinements are shown in Figure 1. The colors indicate the partitioning at each refinement step.

REFERENCES

[1] T. Kvamsdal, R. Holdahl, and P. Böhm. Vista: A Multi-field Object Oriented CFD-package. April 2006.

[2] H. P. Langtangen. *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*. Springer–Verlag, Berlin Heidelberg, 1999.

[3] Diffpack: Software for Finite Element Analysis and Partial Differential Equations. `http://www.diffpack.com`.

[4] ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering. `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`.

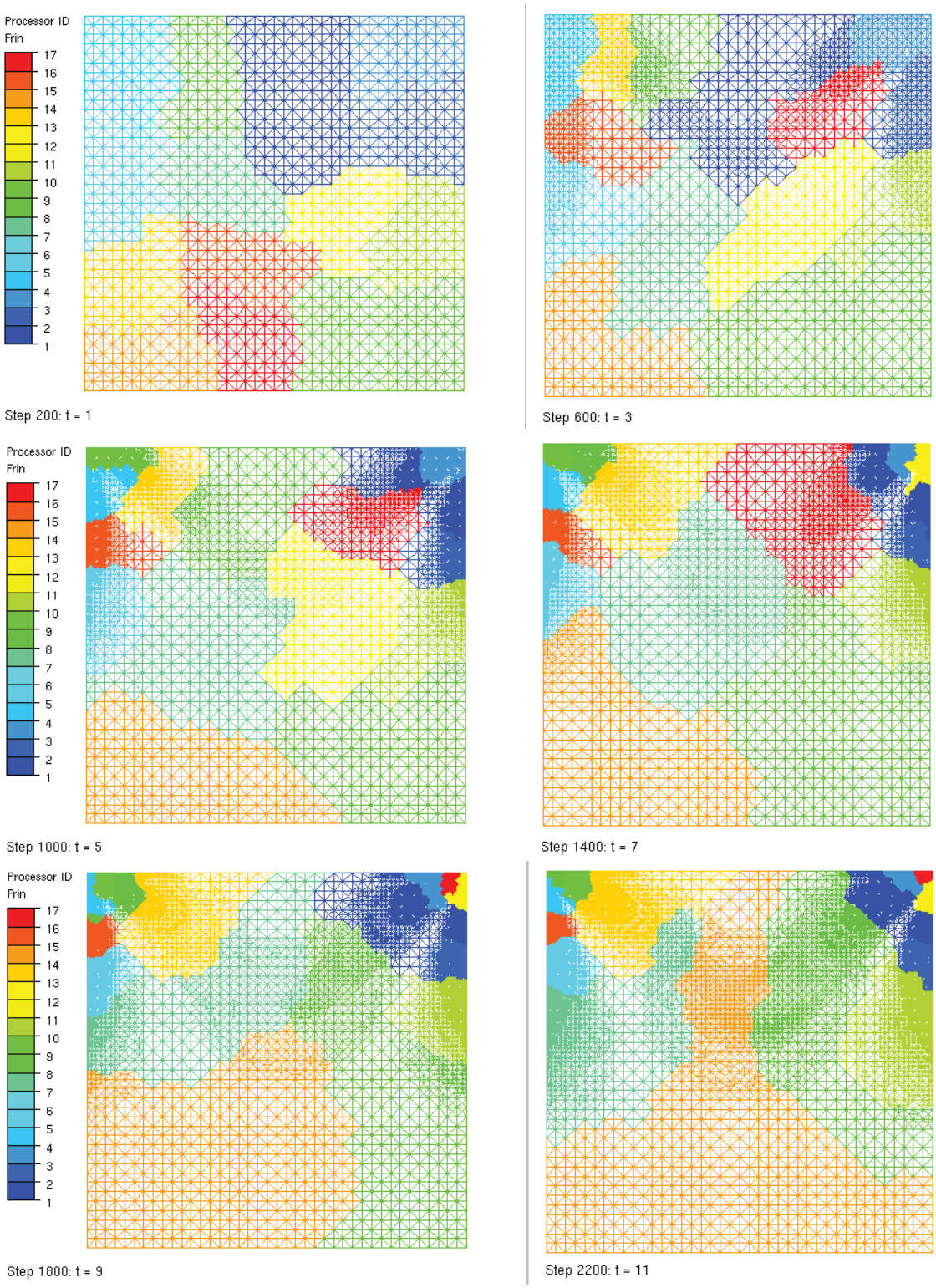


Figure 1. Sequence of adaptively refined grids for the Cavity problem.

Parallel Computation of Incompressible Flow with Cartesian Grid Method

Takahiro Fukushige*, Toshihiro Kamatsuchi*, Yusuke Uda*, Toshiyuki Arima* and Seiji Fujino**

*Honda R&D Co., Ltd. Fundamental Technology Research Center, 1-4-1, Chuo, Wako-shi, Saitama, 351−0193, Japan
(Tel: +81-48-461-2511; e-mail: takahiro_fukushige@n.w.rd.honda.co.jp)
**Kyushu University, 6-10-1, Hakozaki, Higashi-ku, Fukuoka-shi, 812-8581, Japan
(e-mail: fujino@cc.kyushu-u.ac.jp)

Abstract: To solve the flow field around complex geometries is one of the big difficulties for Computational Fluid Dynamics (CFD). To overcome this difficulty, we have already proposed a method. The method consists of two approaches. One is Building-Cube Method and the other is Immersed Boundary method. To reduce the computation time, we adopt MRTR method. The reduction of computational time is obtained by MRTR method.

Keywords: Building-Cube Method, Immersed Boundary Method, Krylov Subspace Method

1. INTRODUCTION

Computational Fluid Dynamics (CFD) has become an important tool for aerodynamics by the improvements of computer performance and CFD algorithm itself. However, the computational time of CFD continues to increase, while progress of computer has been made. One of the reasons is considered that application of CFD has become more complex. For example, CFD is employed to estimate aerodynamics performance for a complex shaped object. Concerning complex shape, however, the problem of grid generation still remains. It requires so much time and labor. To overcome the problems in meshing for complex-shaped object, we develop a method. The method consists of two approaches. One is Immersed Boundary method [1], and the other is Building-Cube Method (BCM) [2]. The basic idea of Immersed Boundary method is applied to cells in the vicinity of solid boundary, and Cartesian grid method is performed for other cells. The method has several advantages. For example, the method is suitable for parallel computation. A large part of computational time of the method is occupied by computation of Poisson's equation for pressure. To reduce the computation time, some latest Krylov subspace methods are implemented. In this paper, the performance improvement of the method is discussed. The parallel efficiency about the parallel computation will be discussed in the final paper.

2. Numerical Methods

2.1 Computational Gird of BCM

The computational grid used in this study is based on the Building-Cube Method. BCM grid generation follows two steps: the first is to generate ‘cube’ of various sizes to fill the flow field as shown in Fig. 1. The second step is to generate Cartesian grid in each cube (Fig. 2). Two cells overlap between adjacent cubes to exchange the flow information at the boundaries.

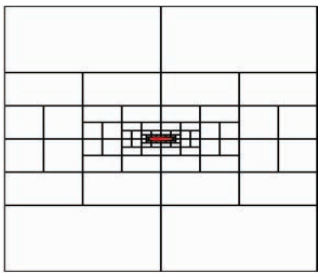


Fig. 1: Cube boundary around airfoil.

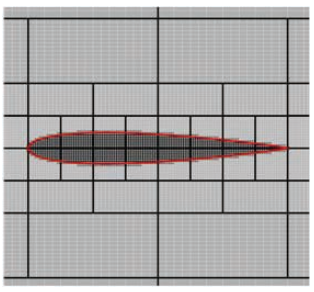


Fig. 2: Cartesian mesh around airfoil.

2.2 Flow Simulation

In this study, mass conservative equation and the Navier-Stokes equations governing incompressible viscous flow are used for solving flow field. It can be written in the Cartesian coordinate system (x, y, z) as

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=0 \tag{1}$$

$$\frac{\partial u\cdot u}{\partial x}+\frac{\partial v\cdot u}{\partial y}+\frac{\partial w\cdot u}{\partial z}=-\frac{1}{\rho}\frac{\partial P}{\partial x}+\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right), \tag{2}$$

$$\frac{\partial u\cdot v}{\partial x}+\frac{\partial v\cdot v}{\partial y}+\frac{\partial w\cdot v}{\partial z}=-\frac{1}{\rho}\frac{\partial P}{\partial y}+\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right), \tag{3}$$

$$\frac{\partial u\cdot w}{\partial x}+\frac{\partial v\cdot w}{\partial y}+\frac{\partial w\cdot w}{\partial z}=-\frac{1}{\rho}\frac{\partial P}{\partial z}+\nu\left(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2}\right), \tag{4}$$

where u, v, w, P,ν and ρ are Cartesian components of the velocity vector in the x, y, z directions, pressure, dynamic viscosity and density, respectively. Specifications of the flow solver in this study are as follows.

- Discretization : FVM
- Convection term : Hybrid Method. (1 upwind, 2nd order central)
- Coupling scheme : Semi-Implicit Method for Pressure-Linked Equations (SIMPLE)

2.3 Parallelization

Equal number of cubes is assigned to each CPU using Message Passing Interface (MPI) library in order to achieve optimal parallel performance. The flowchart of the computation procedure is shown in Fig. 3.

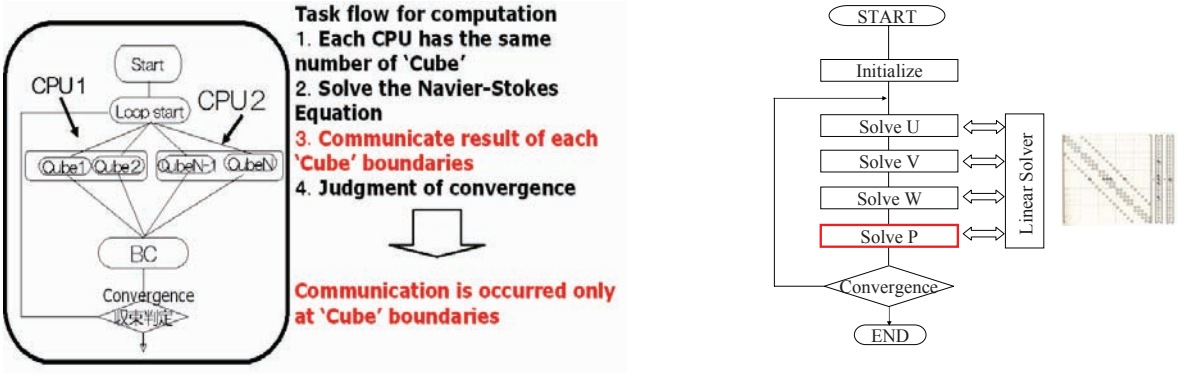


Fig. 3: Flow chart of flow solver (Left: Overall, Right: In each cube).

2.3 Krylov subspace method

In order to efficiently solve the linear equation system obtained from the discretization of Poisson's equation, the following Krylov subspace methods are considered as candidates.

- Minimal Residual Method based on the Three-term recurrence formula of CG type (MRTR) [3]
- Conjugate Residual Squared Method (CRS) [4]
- Conjugate Gradient Squared method (CGS) [5]
- Bi-Conjugate Gradient Stabilized Method (BiCGSTAB) [6]
- Bi-Conjugate Residual Stabilized Method (BiCRSTAB) [4]

3. Computational Result

To evaluate the performance of the above Krylov subspace methods, a series of numerical experiment was carried out for the following two types of model problem, (a) and (b). The test problem is defined as

$$Ax = b, \tag{5}$$

where A and b stand for an n-by-n matrix and an nth order-vector , respectively.

(a) Symmetrical dense matrix:

Coefficient matrix A is defined by

$$A = \begin{bmatrix} N & N-1 & N-2 & N-3 & \cdots & \cdots & 1 \\ N-1 & N-1 & N-2 & N-3 & \cdots & \cdots & 1 \\ N-2 & N-2 & N-2 & N-3 & \cdots & \cdots & 1 \\ N-3 & N-3 & N-3 & N-3 & \cdots & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots & & & 1 \\ \vdots & \vdots & \vdots & \vdots & & & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \tag{6}$$

and right-hand-side vector b is calculated by

$$b = AU, \tag{7}$$

where U is defined by

$$U = [0,1,2\cdots, N-1]. \tag{8}$$

The numerical computations were carried out for N=500, 1000. The convergence history (L2-norm of residual as a function of number of iteration) for MRTR, CRS, CGS, BiCGTAB, BiCRSTAB is shown in Fig. 4. Despite the size of matrix, each method shows similar trend. Specifically, characteristic of MRTR convergence history is favorable, because it doesn't vibrate.

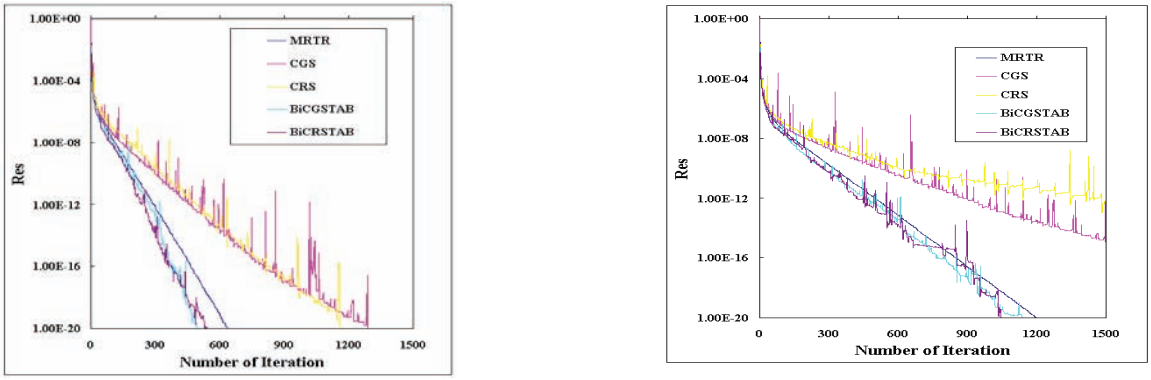


Fig. 4: *Convergence history for MRTR, CGS, CRS, BiCGTAB, BiCRSTAB, with symmetric dense matrix (N=500(left), N=1000 (right)).*

(b) Toeplitz matrix

Coefficient matrix A is defined by

$$A=\begin{bmatrix} 2 & 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 2 & 0 & 0 & \cdots & \cdots & 0 \\ \gamma & 0 & 2 & 0 & \cdots & \cdots & 0 \\ 0 & \gamma & 0 & 2 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & & & 0 \\ \vdots & \vdots & \vdots & \vdots & & & 1 \\ 0 & 0 & 0 & 0 & \gamma & 0 & 2 \end{bmatrix}, \tag{9}$$

and right-hand-side vector b is defined by

$$b=[1,1,1,\cdots 1]. \tag{10}$$

The numerical computations were carried out for N=500, 1000 with γ =1.2. The convergence history for MRTR, CRS, CGS, BiCGTAB, BiCRSTAB is shown in Fig. 5. The convergence of each method except for MRTR depends on the size of matrix.

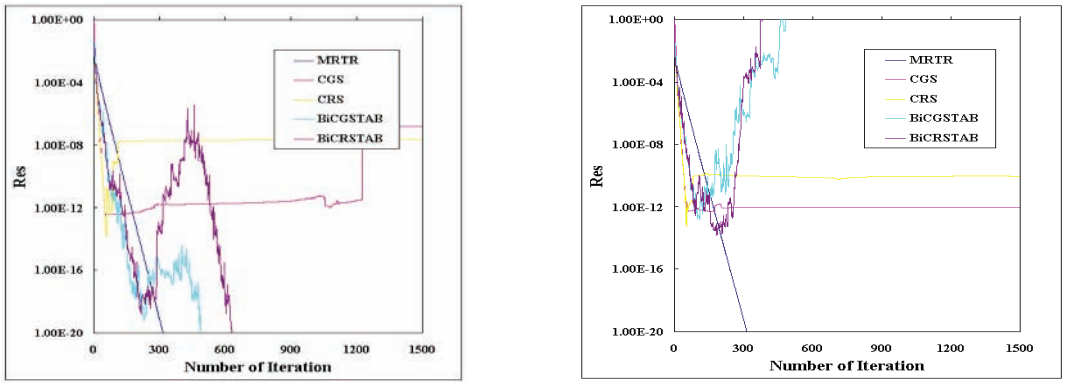


Fig 5: *Convergence history for MRTR, CGS, CRS, BiCGTAB, BiCRSTAB, with Toeplitz matrix (N=500(left), N=1000 (right)).*

To evaluate the performance of the method, flow simulation around sphere was carried out. Computational conditions are shown in Table 1. The surface geometry and computational grid is shown in Fig. 6. One of the examples for computed flow field is shown Fig.7. Large eddy in the wake of sphere was captured in this simulation. This simulation was performed on NEC SX-8. This computer has a vector CPU with 64 GB main memories and 8 processors in each node. Good parallel efficiency is obtained as shown in Fig. 8.

In the final paper, the test result with large number of CPU and convergence history of MRTR method will be discussed.

Table 1: Computational conditions.

Reynolds Number	100
Number of Cube	405
Number of Cell in a cube	16×16×16
Total number of cell	1,658,880

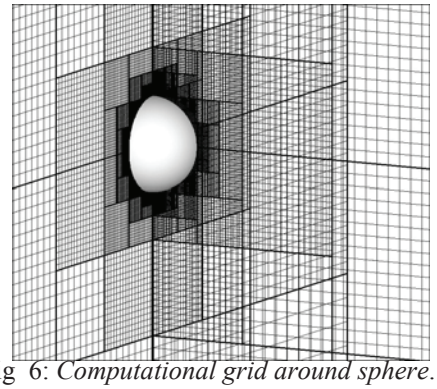


Fig 6: *Computational grid around sphere.*

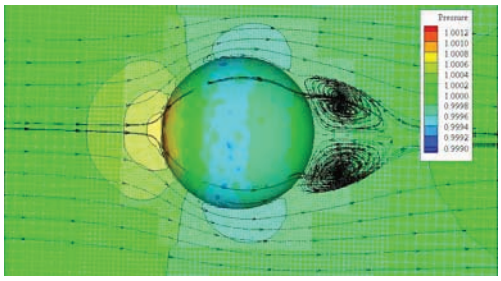


Fig 7: *Computed stream line and pressure contour. (Still under calculation)*

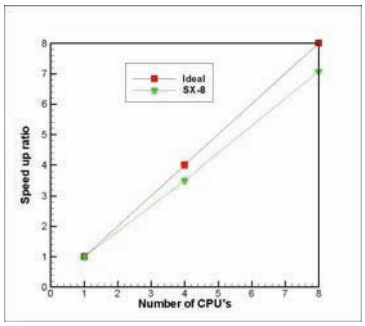


Fig 8: *Parallel efficiency on NEC SX-8*

REFERENCES

1. Peskin,C. S.. (1972). Flow pattern around Hear Valves, J.C.P , Vol.10, pp. 252-275.
2. Nakahashi, K., (2003). Building-Cube method for Flow Problems with Broadband Characteristic Length, Computational Fluid Dynamics, Springer, pp. 77-81.
3. Abe, K., Zhang, S., Mitsui, T., and Jin, C., (2004), A variant of the ORTHIMIN (2) method for singular linear system, Numerical Algorithms, 36, pp. 189-202.
4. Abe, K., Sokabe, T., Fujino, S., and Zhang, S., (2007). A Product-type Krynov Subspace Method Based on Conjugate Residual Method for Nonsymmetric Coefficient Matrices, HPCS2007, pp. 17-24.
5. Sonneveld, P., (1989). CGS, A Fast Lanczos-type Solver for Nonsymmetric Linear System, SIAM J. Sci. Comput., Vol.10, No. 1, pp. 17-24.
6. van der Vorst, H, A., (1992), Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear System, SIAM J. Sci. Stat. Comput., Vol.13, No.2, pp. 631-644.

Parallelization of a Genetic /Artificial Neural Network Based Optimization Algorithm with Application to Low Reynolds Number Airfoils

Miles Bellman*, Brandon Morgan*, Xiaomin Chen* and Ramesh Agarwal*

**Department of Mechanical, Aerospace, and Structural Engineering*
Washington University in St. Louis, St. Louis, MO 63130, USA
(Tel:314-935-6091;email:rka@wustl.edu)

Abstract: In this paper, we employ a genetic algorithm (GA) for shape optimization of low Reynolds number airfoils for generating maximum lift for Unmanned-Air-Vehicle (UAV) applications. The computational efficiency of GA is significantly enhanced with an artificial neural network (ANN). The commercially available software FLUENT is used for calculation of the flow field. It is shown that the combined GA/ANN optimization technique is capable of accurately and efficiently finding globally optimal airfoils.

Keywords: Airfoil Shape Optimization, Genetic Algorithm, Artificial Neural Nets, Unmanned-Air-Vehicles Airfoils, Low Reynolds Number Airfoils

1. INTRODUCTION

Design optimization is a subject of great interest in the aerospace industry since optimization can lead to lighter, faster, and more fuel-efficient aircraft. Due to the highly nonlinear nature of the Navier-Stokes equations, currently almost all practical aerodynamics problems are solved iteratively using the specialized computational fluid dynamics (CFD) software. It is now generally recognized that the traditional gradient-based optimization techniques are difficult to apply and are inefficient when used in conjunction with the Navier-Stokes solvers. Therefore in recent years adjoint methods from control theory [1] and stochastic techniques such as genetic algorithms [2] are increasingly being employed for aerodynamic shape optimization.

In recent years there has been emphasis on the study of Unmanned-Air-Vehicles (UAV) and Micro-Air-Vehicles (MAV). The flow over this class of vehicles is at low Reynolds numbers (< 500,000) in contrast to the Reynolds number of commercial transport aircraft which is in the range of 20 to 40 million. MAV in particular operate on the scale of insect flight (approximately 1.5 to 150 cm wingspan) and thus have very different aerodynamic properties than the typical commercial aircraft. In the commercial aircraft, at high Reynolds numbers, viscous effects are confined to the thin turbulent boundary layer region, and the fluid outside the boundary layer can be accurately approximated as inviscid. In low Reynolds number flow (approximately between Reynolds numbers of 10e3 and 10e5), viscous effects are no longer confined to the thin boundary layer region; they dominate the entire flow field. Therefore the solution of full Navier-Stokes equations becomes a necessity for accurate characterization of the flow fields of UAV and MAV.

In this paper, we describe the application of a genetic algorithm (GA) to optimize low Reynolds number airfoil design. However, since CFD evaluation is computationally expensive and since the GA may require over a thousand evaluations to reach convergence, we additionally build on the work of Carlo Poloni [3] and train an artificial neural network (ANN) online [4] during GA optimization to eventually replace the expensive CFD program for design evaluation. Using this technique, we obtain globally optimal low Reynolds number airfoils over a range of angles-of-attack. The combined GA/ANN is parallelized to run on a SGI Origin 2000 multi-Processor platform.

2. NUMERICAL ALGORITHM

The details of the GA and ANN algorithms are not given here. They are given in References [2] and [4] respectively. Here we briefly describe the implementation of GA/ANN algorithm for airfoil optimization.

2.1 Joukowski Transformation for Airfoil Parameterization

The Joukowski transformation [5] provides an easy mechanism for describing an airfoil by a small number of geometric parameters. It allows us to define an airfoil in the standard coordinate system by transforming a circle in a conformal plane. Let us denote the conformal coordinate system as the ζ-plane with ξ and η axes. The transformation then maps a circle centered in the second quadrant that intersects the positive ξ-axis at point (c, 0) to an airfoil in the z-plane by the following functions:

$$x = \xi\left(1 + \frac{c^2}{\xi^2 + \eta^2}\right) \tag{1}$$

$$y = \eta\left(1 - \frac{c^2}{\xi^2 + \eta^2}\right) \tag{2}$$

This transformation has several desired properties. If the center of the circle is in the second quadrant and the circle intersects the positive ξ-axis, then the airfoil will have a sharp trailing edge. The transformation also ensures that the camber line of the airfoil will start and end on the x-axis, removing the need for any further translations or transformations. Figure 1 illustrates the application of the Joukowski Transformation.

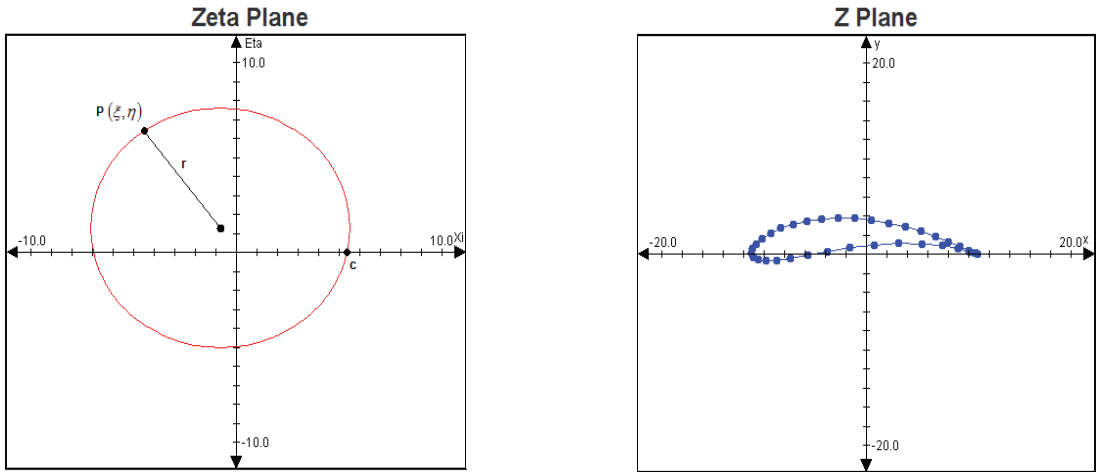


Figure 1: Illustration of the Joukowski Transformation

The thickness of the airfoil t is given by the expression:

$$\frac{t}{c} = -\frac{\sqrt{x_2^2 + x_1^2}}{0.77\left(\sqrt{x_3^2 - x_2^2} + x_1\right)}\cos\left(\frac{\pi}{2} + \tan^{-1}\left(-\frac{x_1}{x_2}\right)\right)$$

where x_1 and x_2 are ξ and η coordinates of the center of the circle, and x_3 is the radius of the circle.

2.2 Algorithm Implementation

In this section we briefly describe the computational setup and the details of the algorithm implementation. Figure 2 schematically illustrates how the GA interfaces with the external mesh generation software GAMBIT [6] and CFD flow solver FLUENT [7]. A GA individual is represented by an airfoil geometry data file, which is passed to the meshing program *Gambit*. Gambit is used to create a two-dimensional structured or unstructured mesh, which is then passed as input to the CFD flow solver FLUENT for computation of the flow field. FLUENT is used iteratively to solve for the coefficient of lift C_l and the coefficient of drag C_d. Some combination of these values (either C_l or C_l/C_d) is taken as the quantity of

interest (objective value) to determine the fitness of the airfoil. The algorithm shown in Figure 2 continues until the convergence in the objective value is achieved. Since a single FLUENT calculation may take as long as thirty minutes to complete on a SGI workstation and since the GA may require over a thousand calculations, we train an ANN online to reduce the runtime. At each step, the ANN's predicted objective value is compared against FLUENT's true objective value, and when the two become close enough to reliably trust the ANN (in practice, this occurs after as few as 200 evaluations), the ANN is used in place of FLUENT for evaluation.

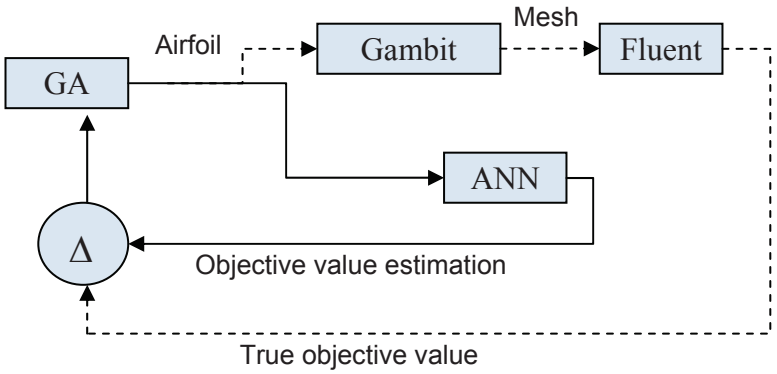


Figure 2: Illustration of information flow in GA process

3. RESULTS AND DISCUSSION

The Joukowski transformation, described in section 2.1, is used to parameterize an airfoil with three parameters: ξ, η, and r. We scale all airfoils to have a chord length of unity and constrain our parameters according to equations (3) given below. These constraints ensure that the airfoils considered are all centered in the second quadrant of the conformal plane, intersect the positive ξ-axis, and have a thickness ratio (maximum thickness to chord length) less than 20%. The constraint on thickness is included to eliminate "fat airfoils."

$$-10 \le \xi \le -0.35\ , 0 \le \eta \le 10\ , \quad \frac{t}{c} \le 0.20 \tag{3}$$

We use a generation size of 10 individuals with a mutation rate of 3% and a natural selection rate of 50% with no culling tolerance; that is, we make no attempt to remove similar airfoils and simply remove the lowest 50% of each generation. The selected individuals are then replaced with an extrapolation-based crossover scheme. Since the objective value can take both positive and negative values, roulette wheel sampling cannot easily be used for selecting reproducing individuals; so, reproduction is done by randomly selecting two individuals. Note, however, that the fittest individuals are still most likely to reproduce because the top 50% of each generation perpetuates to the next generation (and thus will have another chance to reproduce). The offspring individual is then obtained by stepping a random amount in the direction of the fitter parent according to equation (4) below.

$$crossover(\mathbf{x}_1, \mathbf{x}_2) = rand(0,1)\cdot(\mathbf{x}_2 - \mathbf{x}_1) + \mathbf{x}_2 \tag{4}$$

Convergence is determined when the fitness of all individuals in a generation differs by no more than 0.001 (which usually occurs between 150 and 250 generations) or when 250 generations have passed. This second constraint on convergence is applied to prevent the algorithm from running for unreasonable amounts of time.

In this study, our primary objective is to generate optimized low Reynolds number airfoils that maximize C_l at 0° and 2° angles-of-attack. We employ a first-order accurate Navier-Stokes solver on an adaptive mesh with a standard pressure solver in FLUENT. Figure 3 shows the "evolution" of an optimized airfoil using the genetic algorithm at Re = 100,000 and α = 2°. In this figure, we plot the best individual fitness against the number of airfoil evaluations. This figure illustrates the convergence history of the solution during the first 200 airfoil evaluations, which are done using FLUENT. After this point, the ANN is responsible for evaluating the airfoil fitness, and minor modifications continue according to GA until the solution is converged. At completion, the optimal airfoil, as predicted by the ANN, is then evaluated with FLUENT to determine its true objective value. Table 1 and Figure 4 summarize the results obtained for

maximizing C_l using Fluent's first-order accurate Navier-Stokes solver on an adaptive fine mesh. The data in the table shows, as one would expect, that as α increases so does the value of maximum C_l. The C_d values for these optimized airfoils also increases with angle-of-attack. The drag of these airfoils is quite large because the GA has maximized lift without any consideration for drag.

Table 1: Results for airfoils optimized for C_l at Re = 100,000 and α = 0°, 2°

α	Objective	Order of Solution	C_l	C_d	C_l/C_d	Objective Value
0	C_l	1st Order	2.352	0.19007	12.374	2.352
2	C_l	1st Order	3.049	0.18854	16.172	3.049

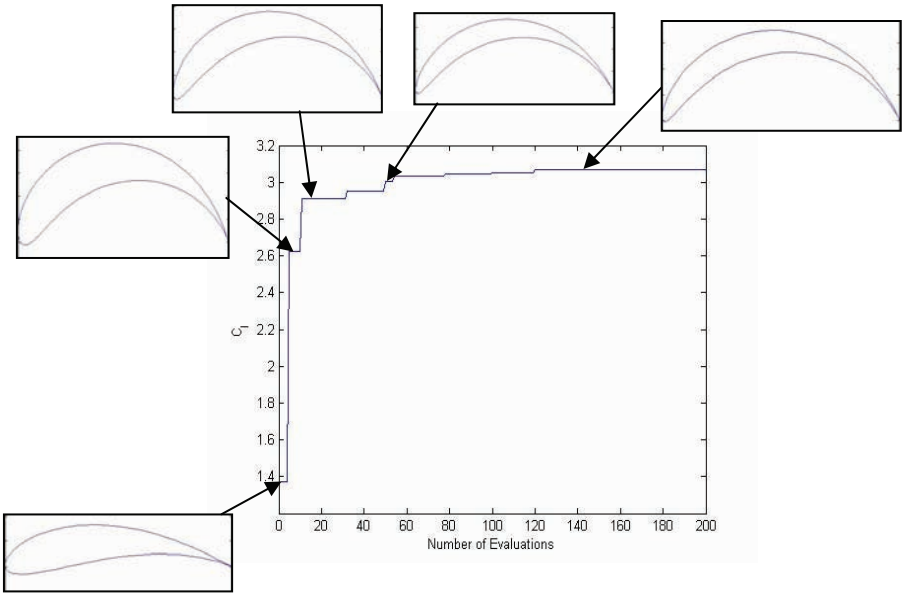


Figure 3: Convergence history of a C_l-optimized airfoil at Re = 100,000 and α = 2°

Figure 4 shows the comparison of the shapes of two airfoils as α increases from 0° to 2°, the optimal airfoil thickness decreases from 11.5% to 10.2% with increase in α and the airfoil becomes more cambered.

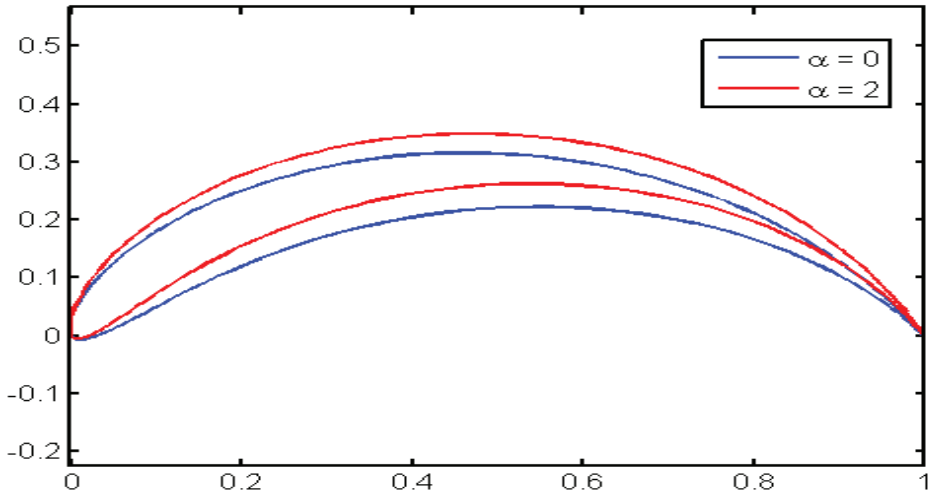


Figure 4: Comparison of the optimized shape of the two airfoils

Figure 5 shows a plot of velocity vectors around the optimized airfoil in Figure 4 at Re = 100,000 and α = 2° while Figure 6 shows the pressure distribution on the airfoil. Neither the velocity vectors nor the pressure distributions in these figures indicate separation bubbles on either the upper or lower surfaces of the airfoils. It is notable that our optimized airfoils do not show flow separation – with or without reattachment. While it is not surprising that airfoils with non-separated boundary layers should outperform those with separation bubbles, it is not obvious that in the flow regime studied, separation could be avoided entirely, as we have seen in our computations.

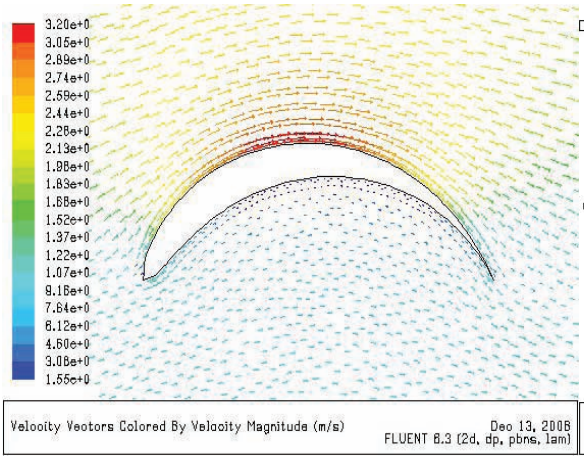


Figure 5: Velocity vectors over a C_l-optimized airfoil at Re = 100,000 and α = 2°

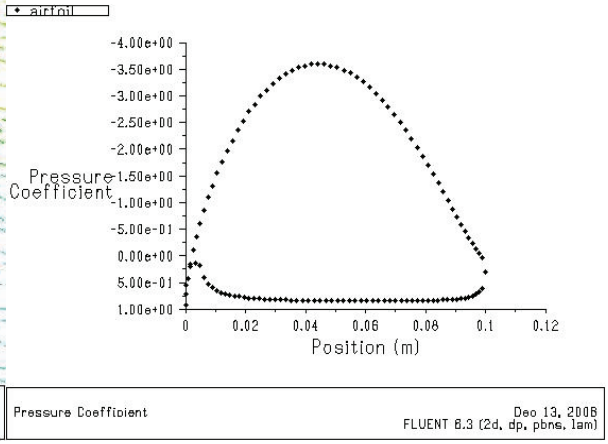


Figure 6: Pressure distribution on C_l-optimized at Re = 100,000 and α = 2°

4. CONCLUSIONS

In this paper, we have employed a genetic algorithm (GA) for shape optimization of low Reynolds number airfoils for generating maximum lift for Unmanned-Air-Vehicle (UAV) applications. The computational efficiency of GA is significantly enhanced with an artificial neural network (ANN). The commercially available software FLUENT is used for calculation of the flow field. It is shown that the combined GA/ANN optimization technique is capable of accurately and efficiently finding globally optimal airfoils. Efforts are currently underway to parallelize the algorithm. A straightforward implementation on 4-processors shows about 90% efficiency.

REFERENCES

[1] Jameson, A. (1988). Aerodynamic design via control theory, *Journal of Scientific Computing*, Vol. 3, pp. 233-260.

[2] Goldberg, D. E. (1989).*Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, New York.

[3] Poloni, C. and Pediroda, V. (1997). GA coupled with computationally expensive simulations: tools to improve efficiency, in *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, John Wiley & Sons, New York.

[4] Alpaydin, E. (2004). *Introduction to Machine Learning*, the MIT Press, Cambridge, MA.

[5] Anderson, John D. (2007). *Fundamentals of Aerodynamics*, 4th ed., McGraw-Hill, New York.

[6] GAMBIT 6.2 (2007). Geometry and Mesh Generation Preprocessor, Ansys Inc.

[7] FLUENT 6.3 (2007). Flow Modeling Software, Ansys Inc.

Propellant Slosh Frequency and Damping Rate Prediction

Brandon S. Marsell*,Dr. Sathya N. Gangadharan**

**Embry-Riddle Aeronautical University, Daytona Beach, FL 32114, USA*
(Tel: 407-492-2065; e-mail: marsellb@erau.edu)
*** Embry-Riddle Aeronautical University, Daytona Beach, FL 32114, USA*
(e-mail: sathya@erau.edu)

Abstract: Energy dissipation and resonant coupling from sloshing fuel in spacecraft fuel tanks is a problem that occurs in the design of many spacecraft. In the case of a spin stabilized spacecraft, this energy dissipation can cause a growth in the spacecrafts' nutation (wobble) that may lead to disastrous consequences for the mission. Even in non-spinning spacecraft, coupling between the spacecraft or upper stage flight control system and an unanticipated slosh resonance can result in catastrophe. By using a Computational Fluid Dynamics (CFD) solver such as Fluent, a model for this fuel slosh can be created. The accuracy of the model must be tested by comparing its results to an experimental test case. Such a model will allow for the variation of many different parameters such as fluid viscosity and gravitational field, yielding a deeper understanding of spacecraft slosh dynamics.

Keywords: Liquid Propellant, Slosh, Frequency, Damping Rate

1. INTRODUCTION

Energy dissipation and resonant coupling from sloshing fuel in spacecraft fuel tanks is a problem that occurs in the design of many spacecraft. In the case of a spin stabilized spacecraft, this energy dissipation can cause a growth in the spacecrafts' nutation (wobble) that may lead to disastrous consequences for the mission [1]. Even in non-spinning spacecraft, coupling between the spacecraft or upper stage flight control system and an unanticipated slosh resonance can result in catastrophe [2]. By using a Computational Fluid Dynamics (CFD) solver such as Fluent, a model for this fuel slosh can be created. The accuracy of the model must be tested by comparing its results to an experimental test case. Such a model will allow for the variation of many different parameters such as fluid viscosity and gravitational field, yielding a deeper understanding of spacecraft slosh dynamics.

The information acquired from the CFD model will be compared to several test scenarios that have been studied in the laboratory in order to verify the results. Once the results have been experimentally verified and there is significant confidence in the values calculated by the CFD technique, it will be applied to various other scenarios such as variable gravitational conditions, larger tanks, different shaped tanks, and more viscous liquids. Ultimately, the model will be modified to include tanks with propellant management devices such as diaphragms and baffles.

2. EXPERIMENTAL SETUP

Experimental data for this research was acquired at Embry Riddle Aeronautical University. The fuel slosh research laboratory is equipped with a state-of-the-art linear actuator and data acquisition system. An eight inch in diameter spherical test tank is suspended from a frame by cables and attached to a linear actuator as seen in Figure 1. A force transducer placed at the interface between the linear actuator and the fuel tank will measure the forces induced by the fuel slosh and transmit the data to a computer for analysis [3].

The effect of energy dissipation in sloshing fuel is best illustrated by exciting the tank with a "sudden push" and quickly bringing it to a stop. This causes the fluid to begin sloshing and slowly damp out the oscillations. The damping effect of the fluid is caused by the fact that the fluid in the tank is not inviscid. The viscous nature of the fluid causes an energy dissipation that will eventually bring the fluid to rest. The maximum amplitude of the reaction force decreases over time. This damping is one of the most important effects that the CFD model must replicate in order to validate the fuel slosh behavior. Another quantity of interest is the frequency at which the slosh oscillations

are occurring. This is useful information that may be used to prevent frequency coupling and inducing resonance in the fuel tank system.

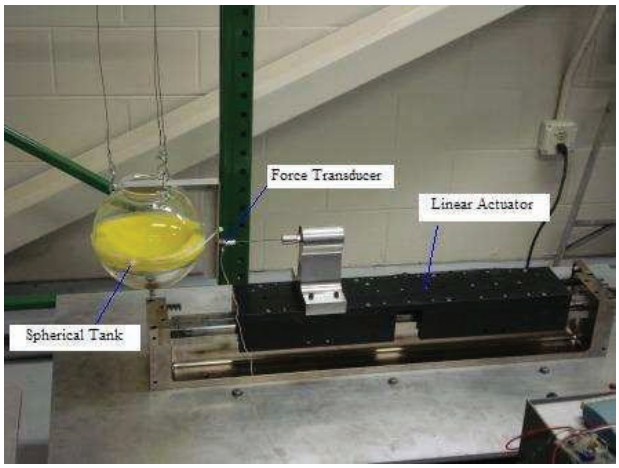


Fig. 1: *Experimental Setup at ERAU*.

3. CFD MODEL

The computational domain used in this study consists of the experimental tank complete with the opening at the top. The tank used is an eight inch spherical tank with a three inch hole at the top that is used for filling. As seen in Figure 2, most of the cells are located at the tank walls. This high resolution at the walls is necessary for capturing the viscous effects that cause the damping of the fluid oscillations. The fully unstructured grid was generated using Gridgen and later modified by Fluent's polyhedral mesh conversion. This polyhedral mesh conversion takes a fully tetrahedral mesh and combines adjoining tetrahedrals to form polyhedrals. The technique reduces the total number of cells without adversely affecting the solution. In this particular problem, the cell number was reduced from 1.2 million to 431,000 therefore greatly reducing the computation time [4].

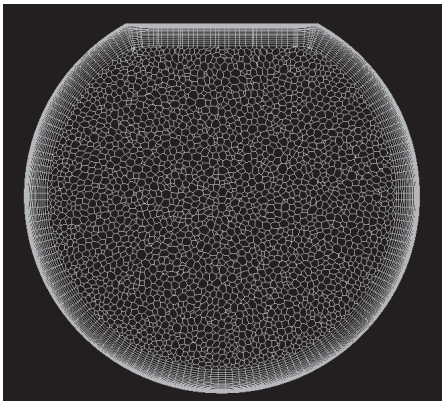


Fig. 2: *Computational Grid*

While solving the Navier-Stokes equations in a CFD calculation is common practice, there is another issued that must be dealt with in this model. The issue that makes this study somewhat more difficult than standard CFD models is that fact that the problem includes two different fluids. When the liquid propellant sloshes around inside the tank, it naturally displaces the air. In order to track the position, shape and velocity of the interface between the liquid propellant and the air, it is necessary to use what is called the volume of fluid (VOF) method [5].

The volume of fluid method developed by Hirt and Nichols introduces a new variable whose value is to be stored at the cell center along with the rest of the solution. This new variable is referred to as the volume fraction. This special volume fraction variable is used as a flag to indicate whether the cell contains fluid or air. For example, if a cell is completely filled with fluid, it will be given a value of unity. On the other hand, if a cell is found in an area that contains air (no fluid) the value for the volume fraction will be zero. If a cell is found to have a volume fraction

value between zero and one, then the cell is said to contain a free surface (Hirt and Nichols). In order to track the value for this volume fraction throughout the entire field as a function of time, Hirt and Nichols define a function F that is to be conserved [5].

$$\frac{\partial F}{\partial t} + u\frac{\partial F}{\partial x} + v\frac{\partial F}{\partial y} + w\frac{\partial F}{\partial z} = 0 \qquad (1)$$

This function F represents the volume fraction at every point in the flow. Notice it is a simple conservation equation much like the continuity equation. The volume fraction at every "node" in a computational domain will either be 0 or 1, but the volume fraction at a cell center can be $0 \leq F \leq 1$. This is due to the fact that the value of F is calculated for the cell center as the average value of its neighboring nodes. For example, as seen in Fig. 3, the two nodes that are not fully submerged in the fluid (green) have a volume fraction of 0 and the two nodes that are fully submerged in the fluid (blue) have a volume fraction of 1. In this two dimensional simplified case, volume fraction at the cell center will be calculated to be 0.5. This concept can be applied to any cell with any number of nodes [5].

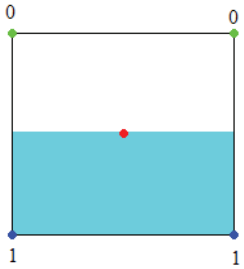


Fig. 3: *Volume Fraction Calculation*

After calculating the volume fraction for each cell center, the next issue that arises is the orientation of the free surface with respect to the cell itself. There are many ways to calculate the orientation of this free surface within the cell, but the most common, and the way it is implemented in this study is by using the geometric reconstruct method. In this method, the free surface is approximated as a line whose volume beneath it is equal to the volume fraction multiplied by the total volume of the cell. The slope of this line is based on the magnitude of the gradient of F (volume fraction) at the faces. This slope will define the shape of the free surface (linear approximation) within the cell. This information allows the solver to track the free surface of the liquid as a function of time [5].

4. PARALLEL PROCESSING

All computations needed for this research were done using the Embry-Riddle University computer cluster. A total of 6 Intel Opteron processors were employed. Since ANSYS Fluent was used as the solver for this study, its automatic partitioning function was utilized. This automatic partitioning made the parallel computing process quite simple saving the researcher a great deal of time.

Parallel computing using ANSYS Fluent makes use of multiple processors by partitioning the domain and data into different sections. Each of these sections is assigned to a single compute node or processor. As each processor is running the computations for their assigned partition it is also communicating with all of the other processors to exchange boundary conditions and interface values. This processor communication process is orchestrated by an ANSYS Fluent utility named cortex. This cortex utility allows fluent to seamlessly integrate the ANSYS Fluent graphical user interface with the complex operations of parallel processing with minimum input from the user. This study is a great testament to the user friendly nature of ANSYS Fluent [6].

5. RESULTS

The damping rate is defined as the rate at which successive peaks in the oscillations diminish in an exponential fit to the data over time. Plotted in Figure 4, are the peaks of the positive oscillations as recorded by the force transducer in the experimental setup. Recall that the tank was given an initial "push" and then allowed to dissipate the oscillations. This peak data, is normalized (to cross the axis at y=1) and then fitted to an exponential curve. This yields a damping rate of 0.0656 for the experimental data shown in Figure 4 [7]. The damping rate was calculated twice for the same tank but using liquids of different viscosities. The two liquids used in the tests were water and glycerin. These two fluids represent a wide range of viscosities that common liquid propellants may fall under.

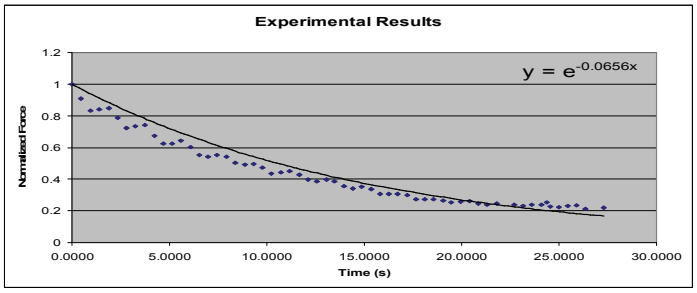


Fig 4: *Experimental Damping Results*

When the tank is given an initial excitation, the liquid in the tank begins to oscillate at its natural frequency. This natural frequency is defined by the gravitational force, fluid fill level, and tank shape. It is very important to be able to predict this natural frequency. If the spacecraft begins to oscillate at this frequency, the forces caused by the sloshing liquid can resonate and greatly amplify causing problems with the control system. The results are outlined in the following table (Table 1).

Table 1: *Results Comparison for Water and Glycerin*

Water		
	Frequency (Primary Mode)	Damping Rate
Experimental	2.120	0.06564
CFD	2.104	0.06557
% Error	0.7	0.11

Glycerin		
	Frequency (Primary Mode)	Damping Rate
Experimental	2.106	1.0833
CFD	2.005	1.1571
% Error	4.80	6.81

Besides accurately predicting both the damping rate and natural frequency, the CFD model can yield valuable information about the fluid as it sloshes. As seen in Figure 5, this model can predict characteristics such as the shape of the free surface and the velocity vector field of the fluid as a function of time. This valuable information helps to visualize the fluid motion, and gain a deeper understanding of the physics behind spacecraft propellant slosh [7].

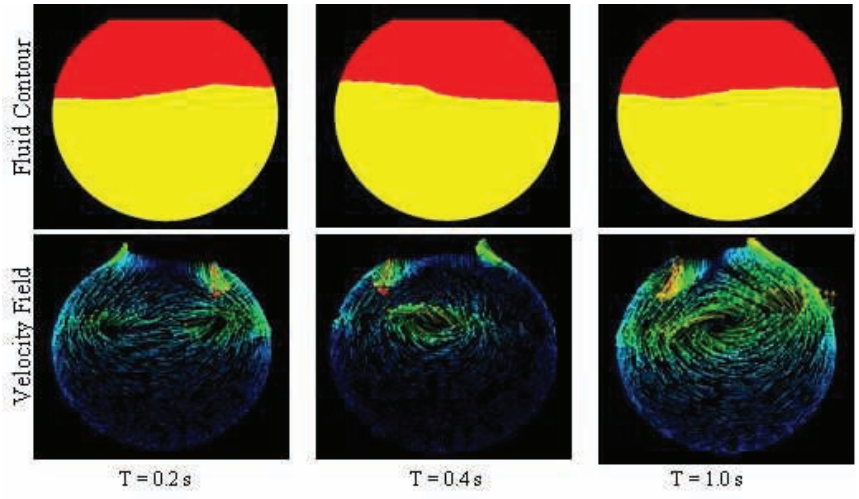


Fig 5: *CFD Results for Fluid Contour and Velocity Vector Field as a function of Time*

6. CONCLUSIONS

This CFD model was successful for simulating the small amplitude slosh of free surface tanks. Further research will be aimed at creating models for more complex environments such as tanks fitted with propellant management devices (PMD's) like baffles and diaphragms. These models will eventually allow the simulation of full scale flight tanks.

Thanks to the power of Computational Fluid Dynamics, an accurate model of spacecraft fuel slosh can be created. This model allows for the prediction of many different parameters that are useful for the completion of any space mission. Besides being an accurate source of data, it allows for a deeper look into the dynamics of sloshing fuel. All of this can be achieved without the cost of building an expensive experimental setup. CFD models like these are the future of fluid studies

REFERENCES

[1] Hubert, C., *Behavior of Spinning Space Vehicles with Onboard Liquids*, Hubert Astronautics, 2nd Edition August 2008.

[2] Musk, Elon. "Updates Archive." SpaceX. 27 March 2007. Space Exploration Technologies. 3 Mar 2009 <http://www.spacex.com/updates_archive.php?page=0107-0707>.

[3] Schlee, K. Gangadharan, S.N., Ristow, J., *Advanced Method to Estimate Fuel Slosh Simulation Parameters,* 41st AIAA Joint Propulsion Conference, 2005.

[4] Gridgen 2007-2008 Pointwise, Inc

[5] Hirt, C.W. and Nichols, B.D., *Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries*, Journal of Computational Physics 39, 201, 1981.

[6] FLUENT Software Version 6.2.16, © 2004 Fluent Inc.

[7] Marsell, B. Gangadharan, S.N. Chatman, Y. et al. *A CFD Approach to Modeling Spacecraft Fuel Slosh* 47th AIAA Aerospace Sciences Meeting, 2009.

Realistic Magnetohydrodynamics Simulations of Turbulent Convection on the Sun

Irina N. Kitiashvili*, Alexander G. Kosovichev, Alan A. Wray***, Nagi N. Mansour******

*Center for Turbulence Research, Stanford University, Stanford, CA 94305, USA
(Tel: 650-723-9596; e-mail: irinasun@stanford.edu)
**W.W. Hansen Experimental Physics Laboratory, Stanford University, Stanford, CA 94305, USA
(e-mail: sasha@sun.stanford.edu)
*** NASA Ames Research Center, Moffett Field, CA 94035, USA
(e-mail: wray@nas.nasa.gov)
**** NASA Ames Research Center, Moffett Field, CA 94035, USA
(e-mail: nagi.n.mansour@nasa.gov)

Abstract: Realistic numerical simulations of the Sun's fluid dynamics and magnetism are very important for the analysis and interpretation of observational data from space missions, such as the Solar and Heliospheric Observatory (ESA/NASA), Solar-B/Hinode (Japan/NASA), and the Solar Dynamics Observatory (NASA), and for developing physics-based methods for making space weather forecasts that are essential for future space exploration, both manned and unmanned. We have developed a highly efficient parallel radiative magnetohydrodynamics code for 3D simulations of the Sun's turbulent convection in magnetic field regions. The code includes all essential physics from first principles, and various subgrid-scale turbulence models. The implementation of this code on the NASA/Ames supercomputer system Columbia shows very efficient, nearly 100% scaling on a large number of processors. The parallelization is done purely with standard MPI methods; no shared-memory techniques were used or needed. The code has been applied for realistic simulations of solar convection and oscillation in the presence of strong inclined magnetic fields. The results reveal very interesting dynamics and self-organization processes, reproducing several phenomena observed in solar active regions, and provide an explanation for the Evershed effect in sunspots.

Keywords: Realistic MHD simulation, subgrid scale turbulence models, solar convection, Evershed effect, solar oscillations.

1. INTRODUCTION

Realistic numerical simulations of the Sun's fluid dynamics and magnetism are very important for the analysis and interpretation of observational data from space missions, such as the Solar and Heliospheric Observatory (ESA/NASA), Solar-B/Hinode (Japan/NASA), and the Solar Dynamics Observatory (NASA), and for developing physics-based methods for making space-weather forecasts that are essential for future space exploration, both manned and unmanned. Large parallel supercomputers, especially those developed in the last few years, have allowed large-scale realistic simulations of solar magnetohydrodynamic convection, and these in turn have greatly increased our knowledge of the structure and dynamics of multi-scale solar convection. A characteristic feature of simulations of this type is that they include all essential physics from first principles, but they do rely on subgrid scale models of turbulence. Jacoutot et al. [1] investigated various turbulence models and showed that the dynamic model of Germano-Moin [3] provides the best agreement with solar observations. These simulations have been used for testing both the local-correlation-tracking and time-distance helioseismology methods for supergranulation flows.

Realistic numerical simulations pioneered by Stein and Nordlund [4] have provided important insights into the structure and dynamics of solar convection and the excitation mechanism of oscillations on the Sun and solar-type stars. These simulations showed the creation of acoustic modes by turbulent pressure and nonadiabatic pressure fluctuations [5]. In this report, we describe a highly efficient parallel radiative magnetohydrodynamics code and simulation results of the Sun's turbulent convection in magnetic field regions. In particular, we investigate the dynamics of convection in regions of a strong, highly inclined magnetic field. The simulation results have provided

important results for interpretation of high-resolution data from the Solar Optical Telescope of the Hinode space mission.

2. RADIATIVE MHD SIMULATION CODE ("SOLARBOX")

The MHD simulation code ("SolarBox") was developed by A. Wray (NASA Ames Research Center). This code allows including various subgrid-scale turbulence models. The code has been carefully tested and has been previously used for studying how various turbulence models affect the excitation of solar oscillations by turbulent convection in the upper convection zone [1, 2]. The implementation of this code on the NASA/Ames supercomputer system Columbia shows very efficient, nearly 100% scaling on a large number of processors (Fig. 1). The parallelization is done purely with standard MPI methods; no shared-memory techniques were used or needed.

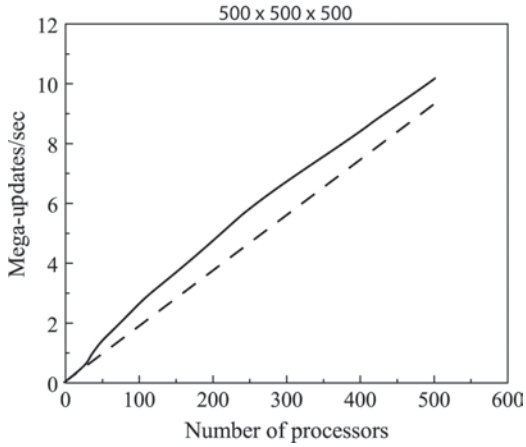


Fig. 1: *Scaling results on Columbia supercomputer for SolarBox MPI code for 500 ^{3} mesh points (solid curve). The dashed line shows linear scaling.*

The code solves the full compressible MHD conservation equations for mass, momentum, energy, and magnetic flux; in tensor notation:

$$\partial\rho/\partial t+\left(\rho u_i\right)_{,i}=0, \qquad \partial\rho u_i/\partial t+\left(\rho u_i u_j+P_{ij}\right)_{,j}=-\rho\varphi_{,i}$$

$$P_{ij}=\left(p+\frac{2}{3}\mu u_{k,k}+\frac{1}{8\pi}B_kB_k\right)\delta_{ij}-\mu\left(u_{i,j}+u_{j,i}\right)-\frac{1}{4\pi}B_iB_j+\rho\tau_{ij},$$

$$\partial E/\partial t+\left(Eu_i+P_{ij}u_j-\left(\kappa+\kappa_T\right)T_{,i}+\left(\frac{c}{4\pi}\right)^2\frac{1}{\left(\sigma+\sigma_T\right)}\left(B_{i,j}-B_{j,i}\right)B_j+F_i^{\mathrm{rad}}\right)_{,i}=0,$$

$$E=\frac{1}{2}\rho u_iu_i+\rho e+\rho\varphi+\frac{1}{8\pi}B_iB_i,$$

$$\partial B_i/\partial t+\left(u_jB_i-u_iB_j-\frac{c^2}{4\pi\left(\sigma+\sigma_T\right)}\left(B_{i,j}-B_{j,i}\right)\right)_{,j}=0,$$

where $\tau_{ij}=-2C_S\Delta^2\left|S\right|\left(S_{ij}-u_{k,k}\delta_{ij}/3\right)+2C_C\Delta^2\left|S\right|^2\delta_{ij}/3$.

Because of the extremely high Reynolds numbers present, direct numerical simulations (DNS) of turbulent motions are not achievable in stellar magnetohydrodynamics. Thus, accurate subgrid-scale models, to describe turbulent stresses in the magnetized plasma, are needed to carry out LES (Large Eddy Simulations). We have carried out numerical experiments to evaluate the performance of different magnetic sub-filter models by means of a priori tests using optimal estimation theory, and their verification with a posteriori tests was done using direct numerical simulations (DNS).

3. EFFECTS OF MAGNETIC FIELD ON TURBULENT CONVECTION

In addition to simulations with initially vertical magnetic fields, we have carried out simulations of solar convection in the presence of inclined magnetic fields. The mean magnetic field inclination is maintained by the top and bottom boundary conditions along with the total magnetic flux. The results presented in Figure 2 show that, in the presence of an inclined field, the magnetoconvection pattern is moving in the direction of inclination similar to the running convective waves.

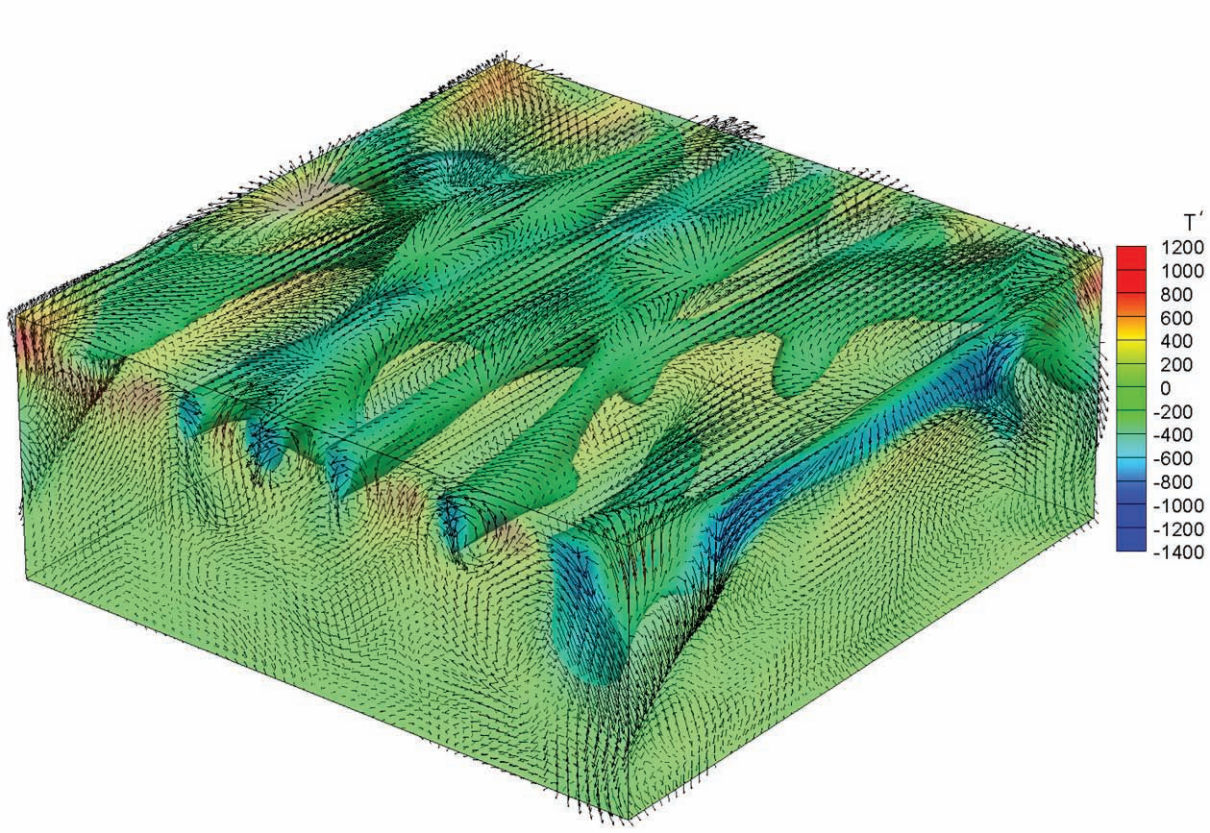


Fig. 2: *Temperature variations (color image) and velocity field in the subsurface convective layer of the Sun, showing the filamentary structure of magnetoconvection and formation of shearing flows caused by traveling convective waves; initial 1200 G magnetic field inclined by 85^0.*

In the realistic 3D case the magnetoconvection develops filamentary structures, and a rapidly moving convection pattern appears (Fig. 2).

4. CONCLUSIONS

We have developed and implemented very efficient methods and parallel codes for magnetohydrodynamic simulations of turbulent convection in the Sun. The realistic simulations of solar convection and oscillation in the presence of magnetic fields reveal very interesting dynamics and self-organization processes, reproducing several phenomena observed in solar active regions. In particular, the results confirm that the spatial scale of granulation substantially decreases with the magnetic field strength. Magnetic field is swept into the intergranular lanes, and the vertical downdraft motions in these lanes are suppressed. This results in a decrease in the excitation power. The oscillation power in the presence of magnetic field is shifted towards higher frequencies, also increasing the amplitude of pseudo-modes above the acoustic cut-off frequency. At moderate field strength of ~ 600 G the power of the high-frequency oscillations reaches a maximum. This corresponds to the phenomenon of "acoustic halo" observed in the range of 5 - 7 mHz at the boundaries of active regions. In the presence of inclined magnetic field the solar convection develops filamentary structures with flows concentrated along the magnetic filaments, and also

exhibits the behavior of running magnetoconvective waves, resembling recent observations of the sunspot penumbra dynamics from the Hinode space mission.

REFERENCES

1. Jacoutot, L., Kosovichev, A.G., Wray, A., and Mansour, N.N. (2008). Numerical simulation of excitation of solar oscillation modes for different turbulent models. *Astrophysical Journal*, 682, 1386-1391.
2. Jacoutot, L., Kosovichev, A.G., Wray, A., and Mansour, N.N. (2008). Realistic numerical simulations of solar convection and oscillations in magnetic regions. *Astrophysical Journal Letters*, 684, L51-L54.
3. Moin, P., Squires, K., Cabot, W., and Lee, S. (1991). A dynamic subgrid-scale model for compressible turbulence and scalar transport. *Phys. Fluids A*, 3(11), 2746-2757.
4. Stein, R.F., and Nordlund, A. (2001). Solar oscillations and convection. II. Excitation of radial oscillations. *Astrophysical Journal*, 546, 585-603.
5. Stein, R.F., Georgobiani, D., Trampedach, R., Ludwig, H.-G., and Nordlund, A. (2004). Excitation of Radial P-Modes in the Sun and Stars. *Solar Physics*, 220, 229-242.

Scalability of Transient CFD on Large-Scale Linux Clusters with Parallel File Systems

Stan Posey*, Bill Loewe, Srinivas Kodiyalam***,
Mark Kremenetsky****, Hugues Mathis*****, Paul Calleja******

** Panasas Inc., Fremont, CA 94555, USA*
(Tel: 510-608-4383; e-mail: sposey@panasas.com)
*** Panasas Inc., Fremont, CA 94555, USA*
(e-mail: bloewe@panasas.com)
**** Silicon Graphics Inc., Sunnyvale, CA 94085, USA*
(e-mail: skodiyalam@sgi.com)
***** Silicon Graphics Inc., Sunnyvale, CA 94085, USA*
(e-mail: mdk@sgi.com)
****** Intel Corporation (UK) Ltd., Pipers Way, Swindon SN3 1RJ, UK*
(e-mail: hughes.a.mathis@intel.com)
******* University of Cambridge, Trinity Lane, Cambridge CB2 1TN, UK*
(e-mail: pjc82@cam.ac.uk*)*

Abstract: This work examines the parallel scalability characteristics of commercial CFD software FLUENT and STAR-CD for up to 256 processing cores, and research CFD software CDP from Stanford University for up to 512 cores – for transient CFD simulations that are heavy in IO relative to numerical operations. In three independent studies conducted with engineering contributions from the University of Cambridge, Intel, SGI, and Panasas, each Linux HPC environment combined Intel Xeon clusters with a Panasas parallel file system and shared storage. The motivation for these studies was to quantify the performance and scalability benefits of parallel IO in CFD software on a parallel file system (PanFS) verses conventional serial NFS file systems for a variety of transient CFD cases.

Keywords: HPC, Transient CFD, Parallel Scalability Linux Clusters, Parallel File Systems, IO.

1. INTRODUCTION

CFD parallel efficiency and simulation turn-around times continue to be an important factor behind engineering and scientific decisions to develop models at higher fidelity. Most parallel CFD simulations use scalable Linux clusters for their demanding HPC requirements, but for certain classes of CFD models, data IO can severely degrade overall job scalability and limit CFD effectiveness. As CFD model sizes grow and the number of processing cores are increased for a single simulation, it becomes critical for each thread on each core to perform IO operations in parallel, rather than rely on the master compute thread to collect and operate on each IO process in serial.

Examples of IO demanding simulations include ~100-million-cell steady state models on a large (> 64) number of cores, and moderate-sized and moderately parallel transient CFD models that require frequent and multiple solution files writes in order to collect time history results for subsequent post-processing. In the case of frequent time history files being written in a serial process, any parallel benefit from a CFD solver is soon overwhelmed as more processing cores (and therefore more serial IO threads) are added to a simulation.

For this reason, commercial CFD software developers such as ANSYS and CD-adapco, and research CFD developers such as Stanford University, offer parallel IO schemes that scale with the CFD solvers in each, but just as parallel solvers require scalable clusters, parallel IO requires that the cluster be configured with a parallel file system and shared storage, and sometimes referred to as parallel network attached storage (NAS). Parallel NAS capability scales IO to overcome IO bottlenecks, enabling IO-bound CFD to scale to its full potential.

2. PARALLEL FILE SYSTEMS AND SHARED STORAGE

A new class of parallel file system and shared storage technology has developed that scales IO in order to extend overall scalability of CFD simulations on clusters. For most implementations, entirely new storage architectures were introduced that combine key advantages of legacy shared storage systems, yet eliminate the drawbacks that have made them unsuitable for large distributed cluster deployments. Parallel NAS can achieve both the high-performance benefits of direct access to disk, as well as data-sharing benefits of files and metadata that HPC clusters require for CFD scalability.

One implementation from Panasas offers a parallel NAS technology with an object-based storage architecture that can eliminate serial IO bottlenecks. Object-based storage enables two primary technological breakthroughs vs. conventional block-based storage. First, since an object contains a combination of user data and metadata attributes, the object architecture is able to offload IO directly to the storage device instead of going through a central file server to deliver parallel IO capability. That is, just as a cluster spreads the work evenly across compute nodes, the object-based storage architecture allows data to be spread across objects for parallel access directly from disk. Secondly, since each object has metadata attributes in addition to user-data, the object can be managed intelligently within large shared volumes under a single namespace.

Object-based storage architectures provide virtually unlimited growth in capacity and bandwidth, making them well-suited for handling CFD run-time IO operations and large files for post-processing and data management. With object-based storage, the cluster has parallel and direct access to all data spread across the shared storage, meaning a large volume of data can be accessed in one simple step by the cluster for computation and visualization to improve speed in the movement of data between storage and other tasks in the CFD workflow. For the Panasas parallel file system and storage, further tuning is developed in shared storage hardware components to optimize the parallel file system software architecture in order to offer an appliance-like implementation.

3. TRANSIENT CFD WITH PARALLEL IO

The combination of scalable CFD application software, Linux HPC clusters and a parallel file system and storage, can provide engineers and scientists with new and significant performance advantages for transient CFD simulations. In recent studies, the advantages of FLUENT 12, STAR-CD v4, and CDP 2.4 with their parallel IO capability, demonstrated efficient scalability, total job turn-around time improvements of 2x and greater, and the ability to conduct high-fidelity yet cost-effective transient solutions that were previously impractical for industrial CFD consideration.

The FLUENT, STAR-CD, and CDP models for these studies comprised cases that were relevant in size and flow features to current commercial and research CFD practice:

- FLUENT *capability* study of a large external aerodynamics case of 111 million cells, modelled as transient with a DES model for performance evaluation from 64 to 256 Intel Xeon cores, and the writing of 20 GB of output data once per every 5 time steps, during 100 iterations.
- STAR-CD transient LES study of a 17 million cell turbomachinery case received from an undisclosed customer for performance evaluation on increasing Intel Xeon cores from 64 to 256, with frequent time history writes of 48 GB of total output data during 300 iterations.
- CDP transient LES study of a 30 million cell internal flow case of a Pratt & Whitney combustor single injector geometry , run on up to 512 cores of an SGI ICE Xeon cluster for scalability evaluation of a 50 iteration restart with a solution write of 12 GB of total data.

FLUENT: Collaboration between ANSYS and Panasas application engineers that began in 2005 has developed a parallel IO scheme based on MPI-IO under the MPI-2 standard. FLUENT parallel IO will be introduced in the FLUENT 12 release scheduled for 1H-2009 and will leverage several parallel file systems including Panasas' PanFS. This development will permit FLUENT scalability for even heavy IO applications such as large steady state models with frequent checkpoints, or even more challenging, transient CFD that may require 100 or more solution writes per single simulation.

The benefits of parallel IO for transient CFD were demonstrated with a production case of a FLUENT aerodynamics model of 111M cells, provided by an industrial truck vehicle manufacturer. Details of the model and results are provided in Figure 1. FLUENT 12 with parallel IO in the way of concurrent writes of local (partition)

solutions to the global solution data file on PanFS demonstrated a ~ 2x performance advantage in total time vs. FLUENT 6.3 on NFS and a conventional NAS system.

In the case of 64 cores, the solver advantage of FLUENT 12 over 6.3 is only 4% and the total time benefit of 1.8x shown in Fig. 1 is a result of parallel IO speed-up. The FLUENT 12 solver advantage grows to 9% at 128 cores, and 24% at 256 cores, which contribute to the growing benefits of total time improvements of 2.0x on 128 and 2.3x on 256 cores for FLUENT 12 on PanFS vs. 6.3 on NFS. It is important to note that the performance of CFD solvers and numerical operations are not affected by the choice of file system, which only improves IO operations. That is, a CFD solver will perform the same on a given cluster regardless of whether a parallel or NFS file system is used.

STAR-CD: This study uses the same STAR-CD v4.6 CFD software on the same cluster, but compares two different file systems. The model and Intel HPC environment details and the results are provided in Fig. 2 and Fig 3. From 64 to 256 cores, solver times are the same regardless of file system, and all the total time gains come from parallel IO speed-up of STAR-CD on PanFS. STAR-CD uses an IO scheme whereby each core writes to its own output file of growing time history results, independent of other cores. The advantage of parallel IO from 64 to 256 cores grows from 24% to 85% and is consistent with results observed in the case of FLUENT 12 from 64 to 256 cores.

CDP: This case also compares the performance of parallel PanFS vs. serial NFS on a serial NAS system for the same CFD software on the same cluster, and for up to 512 cores on an SGI ICE cluster. Details of the model, SGI HPC environment and results are provided in Fig. 4 and Fig. 5. Similar to results observed with the commercial CFD software, CDP improves both overall performance and parallel scalability in this case to the full 512 cores utilized in this study. Total time speed-ups range from a 36% advantage on 128 cores to 82% on 512 cores.

4. CONCLUSIONS

Joint studies conducted between research and industry organizations demonstrate that CFD software with parallel IO on a parallel file system can show full parallel benefit for transient simulations that are heavy in IO relative to numerical operations. The favourable results were conclusive for a range of commercial and research CFD software on a variety of Intel-based Linux HPC clusters with a Panasas parallel file system. Benefits to industry include an expanded and more common use of transient CFD in applications such as aerodynamics, aeroacoustics, reacting flows and combustion, multi-phase and free surface flows, and DES/LES turbulence, among other advanced CFD modeling and practices.

5. FIGURES

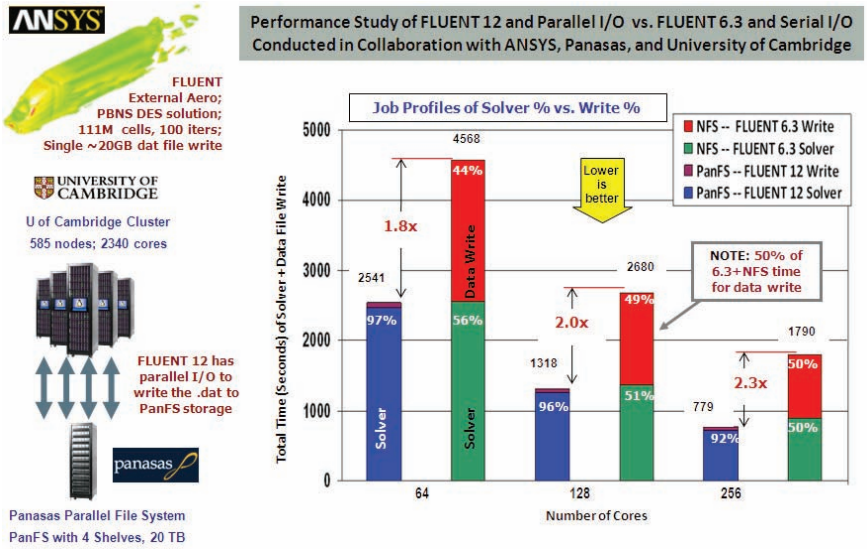


Fig. 1: *Comparison of FLUENT 12 using parallel IO on PanFS vs. FLUENT 6.3 on NFS.*

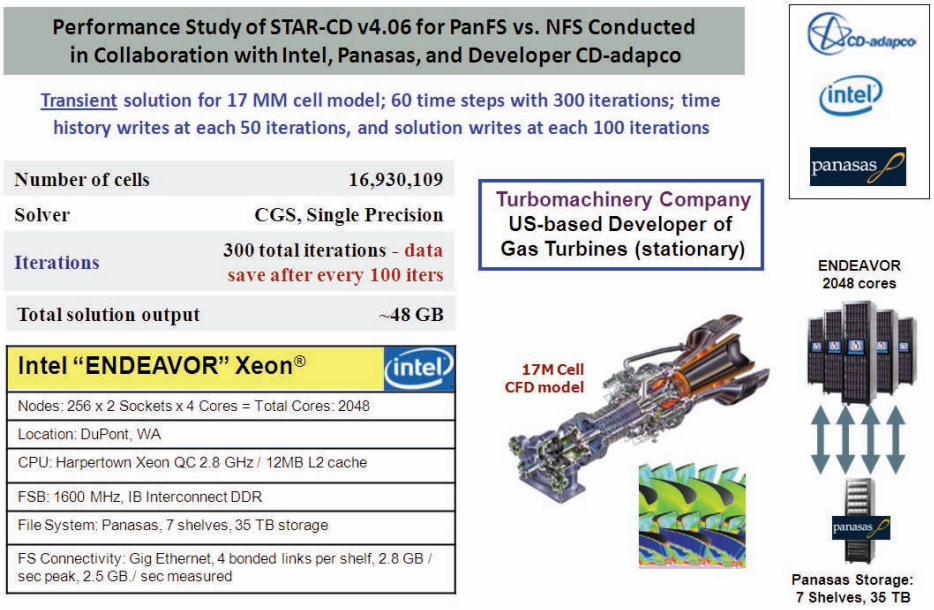


Fig. 2: *Descriptions of the Intel HPC Cluster ENDEAVOR and STAR-CD Input Case.*

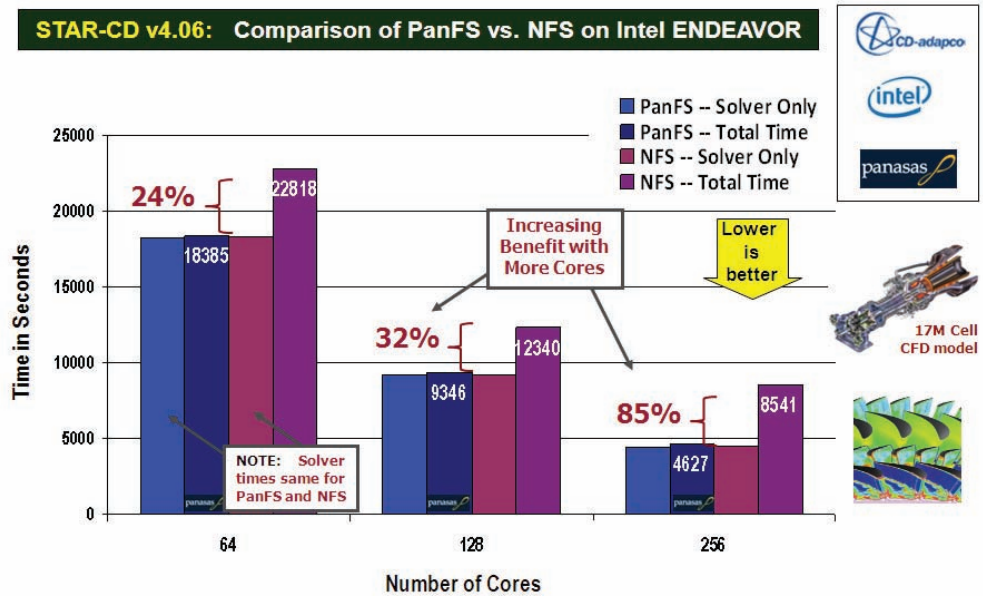


Fig. 3: *Comparison of STAR-CD v4.6 using parallel IO on PanFS vs. serial IO on NFS.*

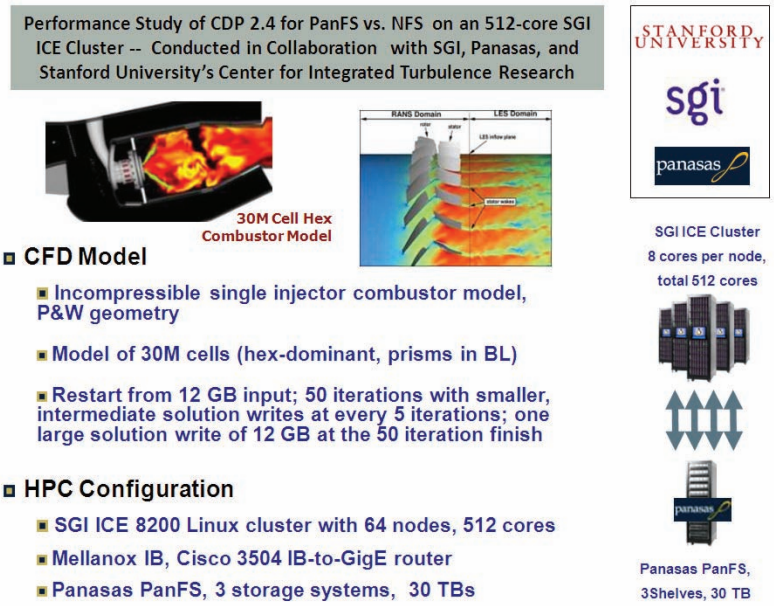


Fig. 4: *Descriptions of the SGI ICE Cluster and CDP Input Case.*

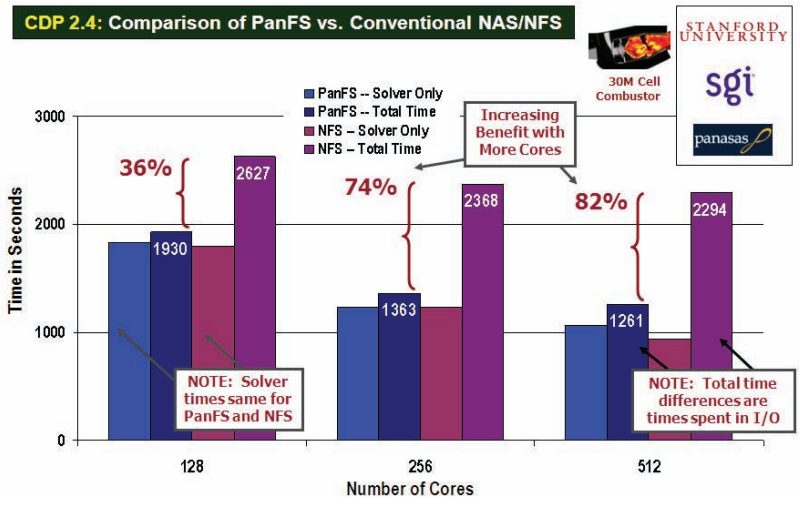


Fig. 5: *Comparison of CDP 2.4 using parallel IO on PanFS vs. serial IO on NFS.*

REFERENCES

1. S. Kodiyalam, M. Kremenetsky and S. Posey, "Balanced HPC Infrastructure for CFD and associated Multidiscipline Simulations of Engineering Systems," Proceedings, 7th Asia CFD Conference 2007, Bangalore, India, November 26 – 30, 2007.
2. Gibson, G.A., Van Meter, R., "Network Attached Storage Architecture," Communications of the ACM, Vol. 43, No. 11, November 2000.
3. FLUENT Software IO Features: http://www.fluent.com/software/fluent/index.htm
4. STAR-CD Software IO Features: http://www.cd-adapco.com/products/STAR-CD
5. Discussions with Dr. Frank Ham, CDP development, Stanford University, Feb – Aug, 2008, and CDP Unstructured LES Code: http://www.stanford.edu/group/cits/research/combustor/cdp.html

Simulation of unsteady incompressible viscous flows using kinetically reduced local Navier-Stokes equations

T. Hashimoto*, K. Morinishi, N. Satofuka*****

**Department of Mechanical Engineering, Kinki University, 3-4-1 Kowakae, Higashi-Osaka, Osaka, 577-8502, Japan*
(e-mail: hasimoto@mech.kindai.ac.jp)
***Department of Mechanical and System Engineering, Kyoto Institute of Technology, Matsugasaki, Sakyo-ku, Kyoto, 606-8585, Japan*
(e-mail: morinisi@kit.ac.jp)
****The University of Shiga Prefecture, 2500 Hassaka-cho, Hikone-shi, Shiga 522-8533, Japan*
(e-mail: satofuka.n@office.usp.ac.jp)

Abstract: The capture of the correct transient behaviour of kinetically reduced local Navier-Stokes (KRLNS) equation for incompressible flow was investigated. The numerical results obtained by the KRLNS equations for a lid-driven 2D square cavity flow and Taylor-Green vortex flow were compared with the artificial compressibility (AC) solutions.

Keywords: Incompressible flow, kinetically reduced local N-S equations, artificial compressibility method

1. INTRODUCTION

Recently, an alternative thermodynamic description of incompressible fluid flows was suggested in the form of kinetically reduced local Navier-Stokes (KRLNS) equation and the capture of the correct time dynamics was studied [1].

In this paper, the numerical simulations of a lid-driven 2D square cavity flow and Taylor-Green vortex flow were presented and the results obtained by the KRLNS equations were compared with the artificial compressibility (AC) solutions.

2. KINETICALLY REDUCED LOCAL NAVIER-STOKES EQUATION

The classical incompressible Navier-Stokes equations consist of the equation for the velocity

$$\partial_t u_\alpha + u_\beta \partial_\beta u_\alpha + \partial_\alpha p = \frac{1}{\mathrm{Re}} \partial_\beta \partial_\beta u_\alpha , \quad (1)$$

and the incompressibility constraint

$$\partial_\alpha u_\alpha = 0 , \quad (2)$$

where $\mathbf{u}$ is the fluid velocity, p is the pressure and Re is the Reynolds number. In the artificial compressibility (AC) method, the time derivative of the pressure is introduced into the continuity equation for coupling between the pressure and the velocity. The continuity equation (2) is written as

$$\partial_t p = -\frac{1}{\delta} \partial_\alpha u_\alpha , \quad (3)$$

where δ is the artificial compressibility parameter. t is an auxiliary variable that can be related to the physical time.

In the form of KRLNS equation, the pressure equation (3) is replaced by

$$\partial_t G = -\frac{1}{Ma^2} \partial_\alpha u_\alpha + \frac{1}{\mathrm{Re}} \partial_\beta \partial_\beta G \quad , \quad p = G + \frac{u^2}{2} \quad (4)$$

where Ma is the Mach number, G is the grand potential. Retaining the term $1/\mathrm{Re}\ \partial_\beta \partial_\beta G$ is crucial for capturing the correct transient behaviour.

3. NUMERICAL RESULTS

The numerical results obtained by the KRLNS equations for a lid-driven 2D square cavity flow and Taylor-Green vortex flow were presented and compared with the AC solutions. In the numerical method, a central difference scheme is used for the spatial discretization and the forward Euler explicit method is used in the time integration. In the case of cavity flow for Re = 400 on a uniform 65×65 Cartesian grid, comparison between KRLNS and AC solutions in the horizontal and vertical velocity profiles and the divergence $\partial_\alpha u_\alpha$ as a function of time are shown in Fig. 1 and Fig. 2, respectively. The adjustable parameters in this case are Ma = 0.1, $\Delta t = 5 \times 10^{-5}$ for the KRLNS and $\delta = 1$, $\Delta t = 10^{-3}$ for the AC. In the case of Taylor-Green vortex flow for Re = 1 on a uniform 33×33 Cartesian grid, comparison of pressure component for KRLNS, AC and the exact solution is shown in Fig. 3. Comparison of divergence as a function of time is shown in Fig. 4. The adjustable parameters are Ma $=10^{-2}$, $\Delta t = 10^{-4}$ for the KRLNS and $\delta = 10^{-4}$, $\Delta t = 10^{-4}$ for the AC. It was confirmed that the KRLNS method can capture the correct transient behaviour due to a smoothing effect of $1/\mathrm{Re}\,\partial_\beta \partial_\beta G$ in Eq. (4), providing that the divergence holds at the zero level, while the divergence in the AC method oscillates.

In the next step, computational efficiency of the KRLNS method is investigated comparing with that of the artificial compressibility method introducing the pseudo time derivative of pressure and velocity, which is solved using the sub-iteration technique.

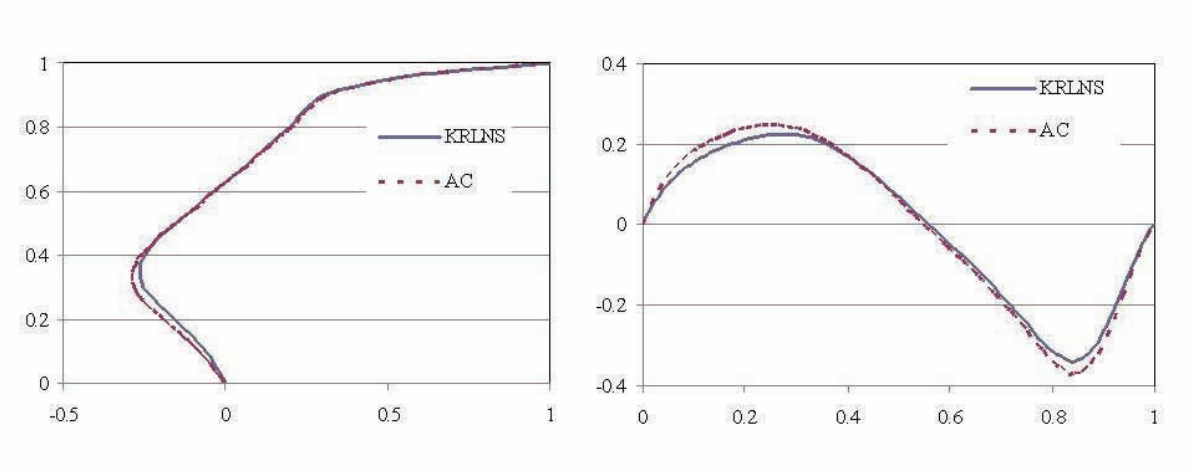


Fig. 1: *Comparison of horizontal and vertical velocity profiles in the cavity flow*.

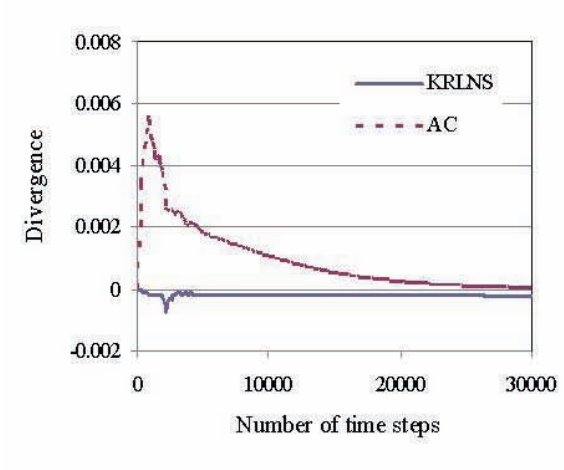


Fig. 2: *Comparison of divergence as a function of time in the cavity flow.*

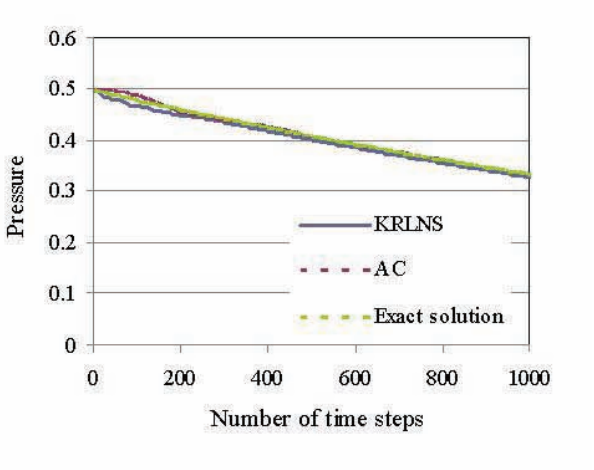


Fig. 3: *Comparison of pressure component in the Taylor-green vortex flow.*

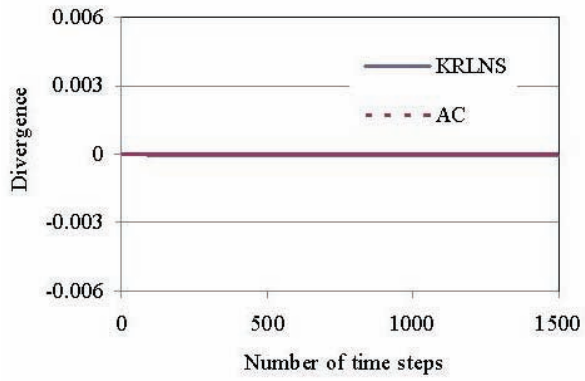


Fig. 4: *Comparison of divergence as a function of time in the Taylor-green vortex flow.*

REFERENCES

1. S. Borok, S.Ansumali and I. V. Karlin. (2007). Kinetically reduced local Navier-Stokes equations for simulation of incompressible viscous flows, *Physical review E 76*, 066704.

SOLVING 3D INCOMPRESSIBLE NAVIER-STOKES EQUATIONS USING FEM AND FRACTIONAL STEP IN PARALLEL COMPUTERS

Alessandro R. E. Antunes[1], Rogério S. Da Silva[2], Paulo R. M. Lyra[3], Ramiro B. Willmersdorf[4]

[1] areantunes@yahoo.com.br, [2] rogsoares@yahoo.com.br, [3] prmlyra@ufpe.br, [4] ramiro@willmersdorf.net,

Federal University of Pernambuco, Departament of Mechanical Engineering
Rua Acadêmico Hélio Ramos, S/N, CEP: 50.740-530, Recife, PE - Brasil

Keywords: *FEM, Incompressible Navier-Stokes, Edge Based Data Structure, Fractional-Step Method, Parallel Computing*

1. Introduction

In this work we developed an implicit monolithic formulation based on a fractional step method for solving the Incompressible Navier-Stokes Equations. The formulation is developed to obtain pressure stabilitys properties and mantain second order precision in time with a stagered solution available (Codina, 2001, Soto *et al.*, 2004). We are using the Finite Element Method and an edge based data structure, which is computationaly atractive. The final algebraic system of equations is solved using PETSc, and the domains partitioning is obtained by ParMetis. Comunications between processors are maneged by MPI. A model application is presented to show the performance of the dsescribed formulation.

2. Mathematical-Numerical Formulation

The incompressible Navier-Stokes Equations, without thermical efects, in continuous form can be writeen like:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} - \upsilon\Delta\mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } \Omega\times(0,t) \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \qquad \text{in } \Omega\times(0,t) \tag{2}$$

where Ω is the spatial fluid domain, t is the time variable, $(0,t)$ is the time interval, $\mathbf{u}$ is the velocity field, υ is the kinematic viscosity, p is the pressure, $\mathbf{f}$ is the external force vector, ∇ is the gradient operator and Δ is the laplacian operator. The Dirichlet and Neuman boundary conditions are:

$$\mathbf{u} = \bar{\mathbf{u}} \text{ in } \Gamma_{du}, \; p = \bar{p} \text{ e } \mathbf{n}\cdot\boldsymbol{\sigma} = \bar{\mathbf{t}} \text{ in } \Gamma_{nu} \tag{3}$$

where $\boldsymbol{\sigma}$ is the viscous stress tensor, $\mathbf{n}$ is the unit outward normal vector, and $\bar{t}$ is the surface stress or traction. Na upper bar refer to a prescribed value.

In which the physical boundary was divided in two non overlaping parts Γ_{du} and Γ_{nu} where the Dirichlet and Neuman boundary conditions are prescribed to each equation, respectivelly. Initial conditions must be known in the whole domain at the initial time.

In this work, a Fractional Step method based in a LU factorization (Codina, 2002, Henriksen *et al.*, 2002, Perot, 1993) was applied. In this method the final system is analogous to the method proposed by Chorin in 1967, and Temam in 1969, that applies a Helmholtz decomposition.

The final discrete system is obtained by using a θ method in time, resulting in a trapezoidal discretization, and a Finite Element Method for the spatial discretization(Gresho *et al.*, 1998, vol 1 & 2). The stabilization of the advective and the gradient pressure terms are obtained with an orthogonal projection of these terms in a finite element space (Codina, 2000, Codina, 2002, Soto *et al.*, 2001).

The variational formulation (Soto *et al*., 2001, Soto *et al*., 2004) can be written as: knowing $\mathbf{u}_h^n$ and p_h^n, find $\left(\mathbf{u}_h^{n+1}, p_h^{n+1}, \boldsymbol{\pi}_h^{n+1}, \xi_h^{n+1}\right)$ in $\mathbf{V}_h\times\mathbf{Q}_h\times\tilde{\mathbf{V}}_h\times\tilde{\mathbf{V}}_h$, such that:

$$\frac{1}{\delta t}\left(\mathbf{u}_h^{n+1,i}-\mathbf{u}_h^{n},v_h\right)+\left(\mathbf{u}_h^{n+1,i-1}\cdot\nabla\mathbf{u}_h^{n+\theta,i},v_h\right)+\left(\upsilon\nabla\mathbf{u}_h^{n+\theta,i},\nabla v_h\right)-\left(p_h^{n+1,i-1},\nabla\cdot v_h\right)$$
$$+\left(\tau\left(\mathbf{u}_h^{n+\theta,i-1}\cdot\nabla\mathbf{u}_h^{n+\theta,i}-\beta^{n+1,i-1}\boldsymbol{\pi}_h^{n+\theta,i-1}\right),\mathbf{u}_h^{n+\theta,i-1}\cdot\nabla v_h\right)=\left(\mathbf{f}^{n+\theta},v_h\right)+\left(\boldsymbol{\sigma}^{n+\theta,i-1}\cdot\mathbf{n},v_h\right)_{\Gamma_{nu}} \quad (4)$$

$$\delta t\left(\nabla p_h^{n+1,i}-\nabla p_h^{n+1,i-1},\nabla q_h\right)+\left(\tau\left(\nabla p_h^{n+1,i}-\beta^{n+1,i-1}\xi_h^{n+1,i-1}\right),\nabla q_h\right)=-\left(\nabla\cdot\mathbf{u}_h^{n+1,i},q_h\right) \quad (5)$$

$$\left(\boldsymbol{\pi}_h^{n+\theta,i},\tilde{v}_h\right)=\left(\mathbf{u}_h^{n+\theta,i}\cdot\nabla\mathbf{u}_h^{n+\theta,i},\tilde{v}_h\right) \quad (6)$$

$$\left(\xi_h^{n+1,i},\tilde{v}_h\right)=\left(\nabla p_h^{n+1,i},\tilde{v}_h\right) \quad (7)$$

$$\forall\left(v_h,q_h,\tilde{v}_h,\tilde{v}_h\right)\in\mathbf{V}_h\times Q_h\times\tilde{\mathbf{V}}_h\times\tilde{\mathbf{V}}_h$$

where $\mathbf{u}_h^{n+1}, p_h^{n+1}, \boldsymbol{\pi}_h^{n+1}, \xi_h^{n+1}$ are the velocity vector, pressure, advective projection term and pressure gradient projection term, respectively, and $\mathbf{V}_h\times Q_h\times\tilde{\mathbf{V}}_h\times\tilde{\mathbf{V}}_h$ are the corresponding finite element functionals spaces, respectivelly. The superscripts *n* and *i* refer to time step and Gauss-Seidel iterations counter in each time step, the subscript *h* refers to discrete variables, δt is the size of time step, β is the stabilization parameter based on pressure gradient, limited to the interval *I=[0 1]*, taking values close to 1 where the pressure field is smooth, and close to 0 in flow regions with sharp pressure gradients (Soto *et. al.*, 2004), τ is the stability and convergence parameter, θ is the time integration parameter (for instance θ = 1.0 Backward Euler e θ = 0.5 Crank-Nicholson), v_h, $\tilde{V}_h$, q_h, are the finite element wheight functions to velocity, advective and pressure gradient projections and pressure, respectivaly. The variables $\boldsymbol{\pi}$ and ξ were obtained by LU factorization of the system. The functional form $(\cdot,\cdot)$, is defined by:

$$\left(a,b\right)=\int_\Omega a\cdot b\,d\Omega\,,\ \left(a,b\right)_\Gamma=\int_\Gamma a\cdot b\,d\Gamma \quad (8)$$

3. Edge Based Data Structure

We are using an edge based data structure, which is advatageous in terms of CPU time, because most of the discrete terms do not need to be re-computed at each iteraction by looping across the elements. The local conservation and symmetry are enforced at the dicrete level calculating only the non-diagonal terms and by computing the diagonal one as the subtraction of the same-row non-diagonal terms (Soto *et al.*, 2004). The final edge based terms are obtained replacing the standard loop over the elements to edge loop, to exemplify, consider the first term of the Eq. (5), which have difusive transport character, and eliminating sub/super-index, yelds:

$$\left(\nabla p,\nabla N\right)=\int_\Omega\nabla p\cdot\nabla N d\Omega=\sum_{E\supset J}\int_{\Omega_E}\nabla N_I\cdot\nabla N_J d\Omega_E\,\hat{p}_J=$$
$$\sum_{S=1}^{m}\left\{\sum_{E\supset IJ}\int_{\Omega_E}\left(\sum_{k=1}^{n\dim}\frac{\partial N_I}{\partial x_k}\frac{\partial N_J}{\partial x_k}\right)d\Omega_E\right\}\left(\hat{p}_I-\hat{p}_J\right)=\sum_{S=1}^{m}C_{IJ}\left(\hat{p}_I-\hat{p}_J\right) \quad (9)$$

where C_{IJ} is associated to edge *IJ*.

Note that the final term of Eq. (9) is naturally conservative, it means that the difusive transport from node *I* to node *J* is equal to the difusive transport from node *J* to node *I*.

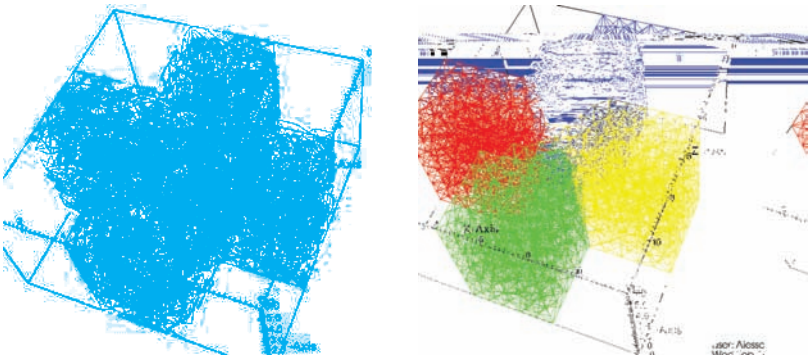
4. Parallel Computing Issues

Problems envolving fluid flow have characteristics extremely complex, and require considerable computational effort. They have non linear, tansient, tridimentional characteristics. In this context, the

use of the parallel computing is necessary to make possible simulations of fuid flows in a satisfactory precision and time.

A parallel computational paradigma in CFD (Computational Fluid Dynamics) is the domain decomposition, that deals with partitioning of the whole domain in a set of non-overlapping subdomains, and each subdomain is associated to one processor. In this work we are using ParMetis (Karypis *et al.*, 2003), which is a parallel library based on MPI (Pacheco, 1997) that implements a variety of algorithms for "optimal" partitioning and re-partitioning applicable to unstructure mesh discretizations. In this case, the comunications between processors are drastically reduced. The partitioning, in this work is made by edges, so that an edge belongs to only one processor and the interfaces are represented by nodes.

Below is presented an example of mesh partitioning.



(a) (b)

Figure 1. Mesh partitioning: (a) initial, (b) with Parmetis.

Figure 1 shows a tridimensional mesh with an initial partitioning, based on a subdivision of the edges by the number of processores, and the same mesh after ParMetis re-partitioning. Note that interfaces were reduced after ParMetis aplication, minimizing necessary comunications between processors. We salient that all parallel algebraic system of equations solved using the PETSc (http:/www-unix.mcs.anl.gov/petsc/petsc-as/documentation), which is a suite of data structures and routines that provide the building blocks for the implementation of large-scale applications.

5. Parallel Tests

In terms of performance of the parallel program many tests were made to verify the efficiency in terms of computational costs reduction. One of these tests refers to speed-up computation, that contrast the relative processing time with number of processors when the number of processors increased.

In this work, processing time for only one processor was obtained by linear extrapolation of the processing time obtained with two processors, because we were not able to perform data pre-processing using only one processor due to lack of memory.

The mathematical expression utilized to determinate speed-up is:

$$S(n,p) = \frac{T_{ref}(n)}{T(n,p)} \tag{10}$$

where n represents the load of work, p is the number of processors, $S(n,p)$ is the speed-up, $T_{ref}(n)$ is the execution reference time obtained with one processor, and $T(n,p)$ is the execution time of the algorithm with p processors.

The numerical example analyzed was the classical lid-driven cavity flow [Ghia *et al.*, 1982]. Figure 2 shows the geometry, where $L = 1$.

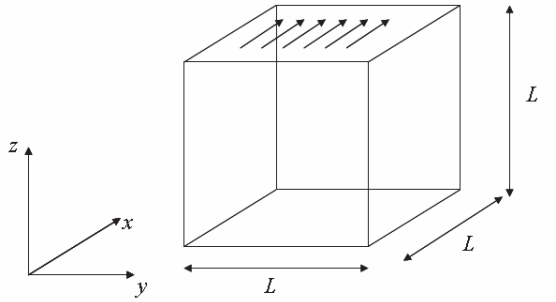
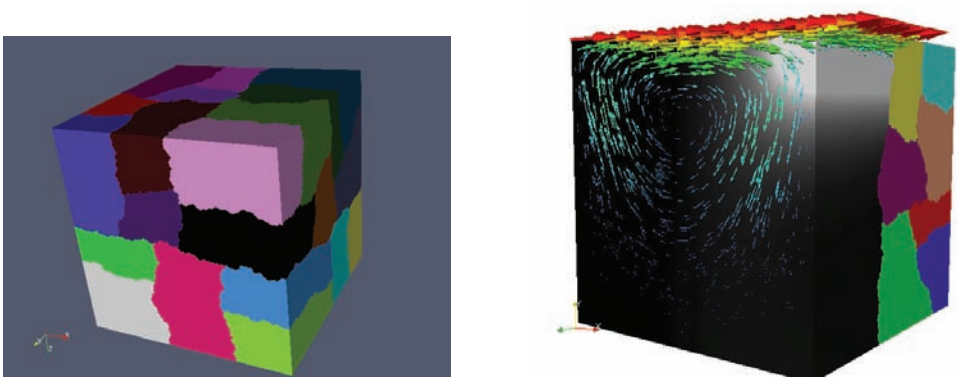


Figure 2. Geometrical and physical domain.

Boundary conditions:

- top: unitary velocity profile prescribed at *x* direction.
- left and right side: viscous tensor prescribed zero at *x* and *z* directions, and velocity component at *y* direction prescribed zero.
- front, back and bottom: non-slip condition.

For *Re = 1,* results were compared with those found in literature with good agreement. The adopted mesh has 1.149.802 nodes, 7.124.082 tetrahedrical elements and 8.398.525 edges, and was partitioned among 2, 4, 8, 12, 16, 20 and 32 processors. Figure (3-a) and Figure (3-b) show the partitions to 32 subdomains and velocity field showing the central vortex, respectively.



(a) (b)

Figure 3. 32 subdomains and velocity field.

The parallel performance is shown in the Fig. (4). Where speedup obtained has been compared with the ideal speedup.

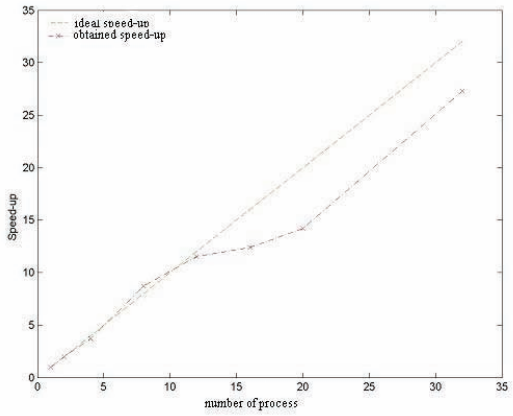


Figure 4. Speedup slope.

We highlight that all simulations were analyzed in the "Núcleo de Atendimento em Computação de Alto Desempenho" (NACAD) of COPPE/UFRJ (http://www.nacad.ufrj.br/), which is a laboratory

of parallel computing applied to engineering and sciences problems, and hold a cluster called Uranus (SGI Altix ICE 8200), with the following configuration:

- 64 CPUs Quad Core Intel Xeon: 256 Colors;
- Memory: 512 Gbytes RAM (distribuited);
- HD Storage: SGI InfiniteStorage NAS (32 TBytes);
- Network: Inifiniband and Gigabit;
- Operational System: Suse Linux Enterprise Server + SGI ProPack;
- Compiles: Intel and GNU (Fortran-90 and C/C++) with OpenMP e MPI support.

6. Conclusions

An implicit monolithic edge based scheme was presented to solve the Incompressible Navier-Stokes Equations. The final mathematical formulation was discretized using Finite Element Method and Fractional Step Method and implemented on parallel computers. This formulation have stabilities properties for the pressure and advective terms based on theirs projections in finite element spaces. The preliminary numerical results indicate that the formulation and implementation is efficient to deal transient incompressible fluid flows. Further investigation are being performed and a more detailed study on the performance of the developed system will be presented soon.

Acknowkedgments

The authors would like to thank the Brazilian Research Council (CNPq) and the Petroleum National Agency (ANP-PRH26) for the financial support and the Center for Parallel Computations (NACAD) for the computational support.

References

Chorin, A. J., 1967, A Numerical Method for Solving Incompressible Viscous Problem, J. Comput. Phys., vol 2.

Codina, R., 2001. Pressure stability in fractional step finite element methods for incompressible flows, J. Computational Physics, vol 170, p. 112-140.

Codina, R., 2002. Stability finite element approximation of transient flows using orthogonal subscales, Comp. Meth. Applied mechanics and Engineering, vol 191, p. 4295-4321.

Gresho, P. M., Sani, R. L., 1998. Incompressible flow and the finite element method, John Wiley & Sons, vols 1 e 2.

Henriksen, M. O., Holmen, J., 2002. Algebraic splitting for incompressible Navier-Stokes equations, J. Computational Physics, vol 175, p. 438-453.

Karypis, G., Schgloegel, K. and Kumar, V., 2003, PARMETIS – Parallel Graph Partitioning and Sparse Matrix Ordering Library, Reference Manual, University of Minnesota

Perot, J. B., 1993. An analysis of the fractional step method, J. Computational Physics, vol 108, p. 51-58.

Pacheco, P., S., Parallel Programming with MPI, Morgan Kaufmam Publishers Inc, 1997.

PETSc Online Manual, Web page. http:/www-unix.mcs.anl.gov/petsc/petsc-as/documentation.

Soto, O., Löhner, R., Cebral, J., An implicit monolithic accurate finite element scheme for incompressible flow problems. In: 15th AIAA Computational Fluid Dynamics Conference, Anaheim, EUA, june, 2001.

Soto, O., Löhner. R., Cebral, J., Camelli, F. 2004. A stabilized edge-based implicit incompressible flow formulation, Comp. Meth. Applied Mechanics and Engineering, vol 193, p. 2139-2154.

Temam, R., 1969, Sur l'approximation de la solutin des équaciones de Navier-Stokes par la méthode dês pás fractionaires (I), Arch. Ration. Mech. Anal., vol 32.

Use of the Cactus Framework for Multi-block CFD Applications

Soon-Heum Ko[1], Prasad Kalghatgi[2], Sumanta Acharya[1),2)], Gabrielle Allen[1),3)], Shantenu Jha[1),3)], Erik Schnetter[1),4)], Mayank Tyagi[1),5)]

[1)] Center for Computation and Technology, Louisiana State University,
216 Johnston Hall, Baton Rouge, LA 70803
[2)] Department of Mechanical Engineering, Louisiana State University,
Baton Rouge, LA 70803
[3)] Department of Computer Science, Louisiana State University,
Baton Rouge, LA 70803
[4)] Department of Physics and Astronomy, Louisiana State University,
Baton Rouge, LA 70803
[5)] Department of Petroleum Engineering, Louisiana State University,
Baton Rouge, LA 70803

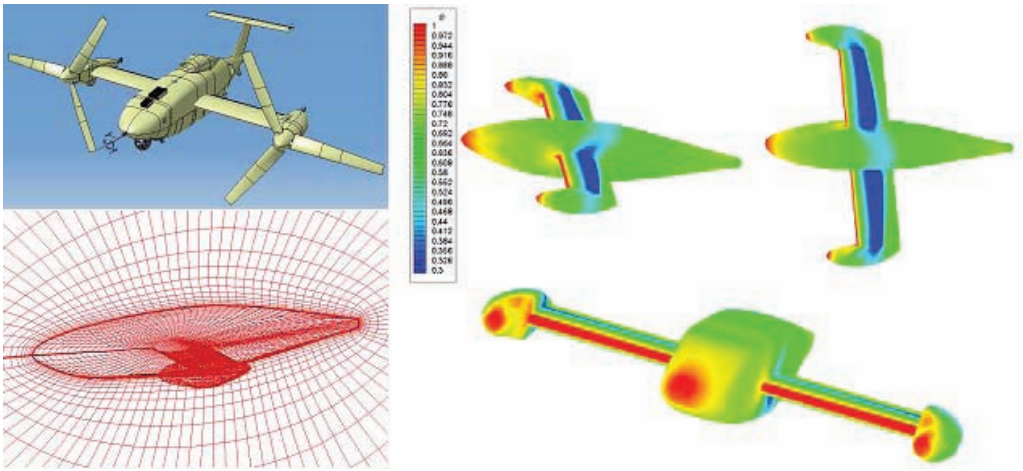
Key Words: Cactus, Multi-block Driver, High Performance Computing

ABSTRACT

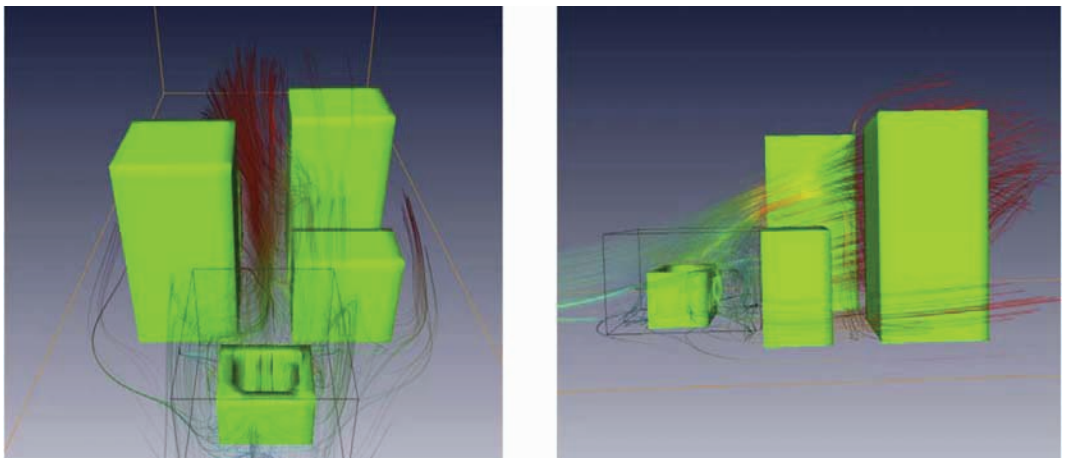
Today, production simulations are routinely deployed on high performance computing resources using parallel codes to resolve complex geometries with multilevel data hierarchies [1]. The modelling of flow is becoming more advanced, with recent research covering diverse velocity variation in a domain, ranging from low-speed to supersonic flow in micro-to-macro sized geometries [2]. Further, CFD research is developing the algorithms necessary for studying multi-physics problems, coupling the effects from other domains such as fluid-structure interaction, aerodynamic-dynamic coupled analysis, and CFD-MD coupling [3]. The development and use of a problem solving environment can be invaluable in addressing modern CFD research topics that inherently involve high-end computation, multi-scale and multi-physics analysis, and interdisciplinary collaboration. First, a framework acts as a gateway to the latest computing infrastructure, including access to supercomputers and use of latest computer scientific drivers/software. Also, frameworks facilitate interdisciplinary and multi-physics collaborative endeavours by providing integrated interfaces, programming standards and modular code development.

As one of a set of general-purpose problem solving environments, the Cactus framework [4] has been used for various scientific simulations since 1997, including astrophysics [5,6] and CFD [7,8,9] Cactus is basically composed of a central core (called the "flesh") and application modules (called "thorns"), which include user-developed scientific code, parallel I/O, checkpointing. Cactus' modular structure enables collaborative code development between different groups, and the central core along with computational tookits supports automatic parallelization, seamless development and deployment on modern computer architectures and easy access to many cutting-edge software technologies such as Globus, HDF5, PETSc library, and advanced visualization tools. In its initial application to the field of CFD, researchers have already successfully conducted parallel computations of compressible or incompressible flow fields on structured domains, by implementing their numerical algorithms, modifying existing Cactus standard modules, and fully utilizing the Cactus infrastructure such as automatic parallelization and use of the PETSc solvers.

However, the initial Cactus-based CFD codes had limitations. Since these codes were developed individually, their application thorns do not follow any programming standard or coordinated development. Thus, the numerical modules employed, such as flux schemes, time evolution routines, I/O interfaces are not easily portable to other applications; boundary condition modules are case-specific. Moreover, current applications have been restricted to single-block geometric problems as seen in Figure 1, due to the absence of multi-block data structure support



a) Compressible CFD Simulation on a Single-block Domain
(Inviscid Analysis on a Simplified Wing-body Configuration at Ref. 8)



b) Incompressible CFD Simulation on a Single-block Domain
(Pollutant Dispersion Simulation on a Cartesian Mesh at Ref. 9)

Figure 1. Cactus-based CFD Applications

From the interest demonstrated thus far in the use of Cactus for CFD applications, and the obvious benefits of shared development of standard components, an initiative has been undertaken to develop a CFD Toolkit within Cactus. This toolkit is being designed to fully utilizes the benefits of the Cactus framework such as numerical library supports and automatic parallelization by drivers, and includes additional drivers and modules specifically developed for CFD applications. New modules include standard I/O routines for CGNS mesh reading and PLOT3D format output, global variable handling on flow (primitive and conservative) variables and geometric variables (mesh points and metric terms), improved data structure supporting multi-block data structure, and general boundary condition modules. Of these elements, some modules such as boundary condition and I/O can be easily implemented by minor modification on existing Cactus thorns, and some are under development now.

A detailed schematic of the Cactus CFD toolkit including a CFD flowchart is shown in Figure 2. Of the CFD modules described on the flowchart, a red solid box indicated users' thorns: initial condition with flow parameter registration, determination of local/minimum time scale and flux schemes. Black dash boxes indicate system-supported modules by function calls inside a Cactus CFD toolkit: mesh reader and coordinate transformation routine will capture geometric features from users' prepared CGNS-format mesh file, boundary conditions will be directly imposed by

user-specified function calls, flow properties automatically updated by declaration on timelevel of variables, and output interface will finally return TecPlot-compatible PLOT3D data. Regarding time integration, users can choose either to use PETSc/Hypre numerical libraries or to adopt their own time integration programming. A multi-block driver along with the Carpet driver will control data structure and parallelization during this simulation, such as registering and managing variables, domain partitioning, and communication control.

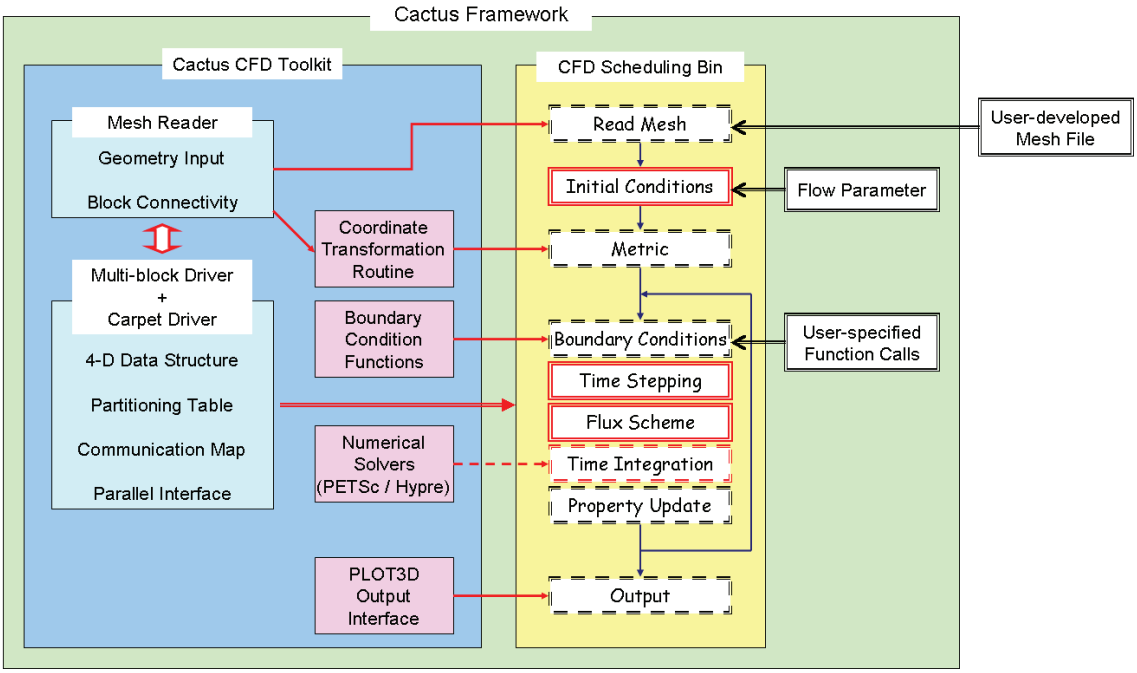


Figure 2. Schematic of a Cactus CFD Toolkit

Incompressible and compressible flow analyses are going to be performed using the resulting Cactus CFD toolkit. An incompressible code under validation uses a cell-entered finite volume method on a block structured curvilinear grid. The transformed Navier-Stokes equations are discretized using a second order upwind-biased scheme for the convective terms, second order central differencing for diffusion terms and a second order Adams-Bashforth scheme for time integration. A fractional step approach is used where an intermediate velocity field is obtained as a first step, a pressure field is then obtained via a pressure Poisson equation and the intermediate velocity field updated for the pressure gradient. To solve for the intermediate velocity field, a hybrid formulation on a curvilinear grid [10] is used where all field variables and pressure are stored at the cell-center collocated location, but the velocity field is updated at the staggered cell-face locations. Currently the Hypre linear solver library is being used to solve system of linear equations in parallel. A number of validation studies have been performed to demonstrate the convergence and accuracy of the approach. A compressible Cactus code has been developed and used for a number of applications, and details on numerical approaches are also described in Refs. 7 and 8.

ACKNOWLEDGEMENT

The current work is funded by the NSF EPSCOR CyberTools project under award #EPS-0701491.

REFERENCES

[1] G. A. Richardson, W. N. Dawes, A. M. Savill, (2007). "An Unsteady, Moving Mesh CFD Simulation for Harrier Hot-Gas Ingestion Control Analysis", *The Aeronautical Journal*, Vol.111, No. 1117, pp.133-144

[2] D. You, P. Moin, (2008). "Active Control of Flow Separation over an Airfoil using Synthetic Jets", *Journal of Fluids and Structures*, Vol.24, No.8, pp.1349–1357

[3] C. H. Tai, K. M. Liew, Y. Zhao, (2007). "Numerical Simulation of 3D Fluid–Structure Interaction Flow using an Immersed Object Method with Overlapping Grids", *Computers and Structures*, Vol.85, No.11-14, pp.749-762

[4] http://www.cactuscode.org

[5] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, J. Shalf, (2001). "The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment", *International Journal of High Performance Computing Applications*, Vol.15, No.4, pp. 345-358

[6] G. Allen, T. Goodale, G. Lanfermann, T. Radke, E. Seidel, W. Benger, H. Hege, A. Merzky, J. Massó, J. Shalf, (1999). "Solving Einstein's Equations on Supercomputers", *IEEE Computer*, Vol.32, No.12, pp. 52-58

[7] S.-H. Ko, K. W. Cho, C. Kim, J. Na, (2007). "Design of CFD Problem Solving Environment based on Cactus Framework", in *Parallel Computational Fluid Dynamics 2006: Parallel Computing and its Applications*, Edited by J.-H. Kwon, J. Periaux, P. Fox, N. Satofuka, A. Ecer, Elsevier, pp.165-172

[8] http://www.ksiam.org/conference/annual062/upfile/Cactus%20for%20KSIAM_1.pdf

[9] http://www.cactuscode.org/News/RazvanCarbunescuLAS2007

[10] L. Ge, F. Sotiropolous, (2007). "A Numerical Method for Solving the 3D Unsteady Incompressible Navier-Stokes Equations in Curvilinear Domains with Complex Immersed Boundaries", *Journal of Computational Physics*, Vol.225, No.2, pp.1782-1809

CONFERENCE TUTORIAL

Conference Tutorial: Hybrid OpenMP/MPI Programming and Other Models for Multi-Core Architectures

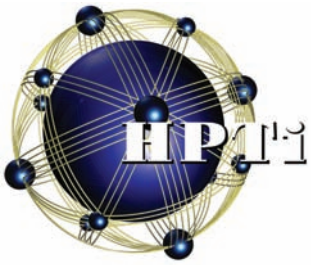
- *Gabriele Jost, Texas Advanced Computing Center, The University of Texas at Austin*
- *Alice Koniges, NERSC, Lawrence Berkeley National Laboratory*
- *Gerhard Wellein and Georg Hager*, Erlangen Regional Computing Center, University of Erlangen-Nuremberg, Germany*
- *Rolf Rabenseifner,* High Performance Computing Center Stuttgart*

Multi-core architectures with an increasing number of cores are beginning to dominate compute platforms for scientific and engineering simulations. One potential means of programming these systems includes combining MPI with OpenMPI in a hybrid model. In this tutorial, we describe the ways in which the hybrid model can be programmed, and discuss its performance compared with pure MPI. We also review current multi-core architectures and provide an introduction to alternate languages such as Co-array Fortran (CAF) and Unified Parallel C (UPC). We present benchmark results on several platforms and identify which applications are likely to benefit from hybrid programming models.

*author only—not speaking

CONFERENCE SPONSORS

A heartfelt thank you to the sponsors of Par CFD 2009:



AUTHOR INDEX

AUTHOR INDEX