

LNCS 5976

Douglas J.K. Mewhort
Natalie M. Cann
Gary W. Slater
Thomas J. Naughton (Eds.)

High Performance Computing Systems and Applications

**23rd International Symposium, HPCS 2009
Kingston, ON, Canada, June 2009
Revised Selected Papers**

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Germany

Madhu Sudan

Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Douglas J.K. Mewhort Natalie M. Cann
Gary W. Slater Thomas J. Naughton (Eds.)

High Performance Computing Systems and Applications

23rd International Symposium, HPCS 2009
Kingston, ON, Canada, June 14-17, 2009
Revised Selected Papers



Springer

Volume Editors

Douglas J.K. Mewhort
Queen's University, Dept. of Psychology
62 Arch St, Kingston, ON, K7L 3N6, Canada
E-mail: mewhortd@queensu.ca

Natalie M. Cann
Queen's University, Dept of Chemistry
Chernoff Hall, Kingston, ON, K7L 3N6 Canada
E-mail: ncann@chem.queensu.ca

Gary W. Slater
University of Ottawa
Hagen Hall, 115 Séraphin-Marion
Ottawa, ON, K1N 6N5 Canada
E-mail: deangrad@uottawa.ca

Thomas J. Naughton
Oak Ridge National Laboratory
1 Bethel Valley Road, Bldg. 5100
Oak Ridge, TN, 37831-6173, USA
E-mail: naughtont@ornl.gov

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D, F.1.2, C.1.4, C.2.4, B.2.1, H.4-5, H.2.8

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-12658-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-12658-1 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2010
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

This book reports the scientific proceedings of the 23rd International Symposium on High Performance Computing Systems and Applications. HPCS 2009 was held at Queen's University at Kingston, June 14–17, 2009. The symposium was hosted by HPCVL in co-operation with Compute Canada.

The symposium brought together over 150 scientists from many countries including Brazil, Canada, China, UK, Germany and the USA. As the proceedings demonstrate, their work addresses fundamental HPC-related issues, from programming, to architectures, to applications and, finally, to upcoming challenges for HPC.

In addition to the scientific program, reported here, the conference was addressed by keynote speakers on various topics including:

- Andy Bechtolsheim, Sun Microsystems and Arista. “The Road from Peta-Flop To Exa-Flop: Exploring How Technology Road-Maps over the Next Ten Years Will Enable us to Build a Sustained Exa-Flop Computer by 2020.”
- Russ Boyd, Alexander McLeod Professor of Chemistry, Dalhousie University. “Computational Chemistry: History, Grand Challenges and Opportunities.”
- André Bandrauck, Canada Research Chair, Computational Chemistry and Molecular Photonics, Université de Sherbrooke. “MAST–Molecular Attosecond Simulations and Theory.”
- Wolfgang Gentzsch, DEISA Project. “The European e-Infrastructure DEISA for e-Science Grand Challenges.”
- Cynthia McIntyre, Sr. V.P. USA Council on Competitiveness. “How HPC Will Improve Competitiveness in the US.”
- Tony Noble, Queen's University, Director of SNOlab. “HPC and the Sudbury Neutrino Laboratory.”
- Suzanne Fortier, President, Natural Sciences and Engineering Research Council of Canada. “HPC and the Natural Sciences and Engineering Research Council.”

HPCS2009 is grateful to many people. We want to thank the symposium's sponsors: SUN Microsystems, Dell, IBM, Cray, Altair, Tycrid Platform Technologies, DataDirect Networks, SGI, O3 Collaboration, NAG, Microsoft, KEDCO, HPC Wire.

The editors thank the many people who helped organize the conference including:

- Conference Chair: Ken Edgecombe (Executive Director, HPCVL)
- Program Chair: Doug Mewhort
- The Scientific Committee:
 - Doug Mewhort (Psychology, Queen's University)
 - Ugo Piomelli (Mechanical & Materials Engineering, Queen's University)
 - Natalie Cann (Chemistry, Queen's University)

- Ahmad Afsahi (Electrical Engineering, Queen's University)
 - Selim Akl (Computing Science, Queen's University)
 - Frank Dehne (Computing Science, Carleton University)
 - Joerg Sack (Computing Science, Carleton University)
 - Jean Fugere (Military Modelling, Royal Military College)
 - Stavros Tavoularis (Mechanical Engineering, University of Ottawa)
 - Gary Slater (Biological Physics, University of Ottawa)
 - Kamran Behdinan (Aerospace Engineering, Ryerson University)
 - Thomas Naughton (Oak Ridge National Lab)
- Conference Administration: Ashley Wightman
 - Workshop Coordinator: Hartmut Schmider
 - Technical and Web support: Michael Hanlan

We also thank the many referees who worked hard to help us select the papers for publication and who offered sound advice to the authors. Finally, we thank all symposium participants and, of course, those who submitted papers for the conference proceedings.

Conference Workshops

The scientific program was extended by six workshops on various technical aspects of HPC. The workshops were:

1 Using OpenMP 3.0 for Parallel Programming on Multicore Systems

Ruud van der Pas, *Sun Microsystems*; Dieter an Mey and Christian Terboven, *Technical University of Aachen*

Abstract. The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ and Fortran. OpenMP is a portable, scalable model with a simple, flexible interface for developing parallel applications on platforms from desktop to supercomputer. In May 2008, OpenMP Version 3.0 was finalized, including the new tasking concept as the major new feature. Tasking considerably increases OpenMP's applicability to a wider class of applications. Furthermore, the support for nested parallelization was substantially enhanced. The workshop presented a basic introduction to OpenMP, performance aspects of OpenMP programming, and case studies demonstrating performance tuning with OpenMP.

2 HPC Meets Dtrace: A Getting Started Tutorial for Dtrace Newbies

Thomas Nau, *University of Ulm*

Abstract. To handle performance bottlenecks, many developers, researchers, and system administrators worry primarily about CPU cycles, caches or compiler flags. Too often the influence of the operating system or of interactions between concurrent application is neglected. Such neglect reflects a paucity of easy-to-use tools that cover applications in both user- and kernel-space. With the introduction of DTrace in Solaris 10, a comprehensive tool is now available. The tutorial targeted new users of DTrace from both a program development and a system-administration perspective. It focused on how Dtrace works and on how easily information from an application or from the Solaris kernel can be retrieved. Examples included a broad range of issues, from IO behavior to in-depth thread scheduling.

3 Using the Pilot Library: A Fresh Alternative to MPI for HPC Clusters

William Gardner, John Carter and Natalie Girard, *University of Guelph*

Abstract. *Pilot* is a new way to program high-performance clusters based on a high-level model featuring processes executing on cluster nodes, and channels for passing messages among them. Designed to smooth the learning curve for novice scientific programmers, the set of library functions is small—less than one-tenth that of MPI—and easy to learn, since the syntax mirrors C's well-known printf and scanf. The

process/channel abstraction inherently reduces the opportunities for communication errors that result in deadlock, and a runtime mechanism detects and diagnoses deadlocks arising from circular waiting. The Pilot library is built as a transparent layer on top of conventional MPI, and shields users from the latter's complexity while adding minimal overhead.

4 Using the NAG Numerical Libraries on Multicore Systems

Jeremy Walton, *Numerical Algorithms Group*

Abstract. Multi-core is the roadmap for all future processors; it provides the opportunity for high performance through the use of multiple cores working together in parallel. However, software must be parallelized to make use of the multiple cores. This workshop introduced the NAG Numerical Library and described the way that it has been optimized for multicore systems. The NAG Numerical Library underpins thousands of applications used world-wide in fields such as finance, science, engineering, academia, and research. Since its first release more than 35 ago, NAG has been widely trusted because of its unrivalled quality, reliability and portability. The workshop began with a discussion of the pitfalls that await the unwary in numerical computation. It introduced the NAG Numerical Library as a comprehensive collection of mathematical and statistical algorithms for programmers and application developers. The routines are accessible from many environments, including standard languages such as Fortran, C and C++, as well as packages such as MATLAB, Maple and Excel. The routines provide a straightforward way to incorporate powerful, reliable and accurate algorithms into applications. The workshop highlighted those routines that have been specially developed and tuned for use on multi-core systems and other shared -memory architectures.

5 Parallel Programming in Visual Studio 2008 on Windows HPC Server 2008

Christian Terboven, *Technical University of Aachen*

Abstract. Visual Studio 2008 is a powerful integrated development environment for C/C++ programmers. It provides comfortable tools for code design and editing tasks, as well as advanced debugging capabilities for serial, multi-threaded and message-passing programs. Windows HPC Server 2008, Microsoft's second incarnation of a complete HPC cluster solution, includes a well-integrated MPI library and a batch system. This environment is completed by ISVs such as Allinea (providing the Visual Studio plugin, DDTlite, to enhance the debugging capabilities for parallel programs) and Intel (providing Parallel Studio to allow for easy-to-use performance analysis and correctness-checking of serial and parallel programs as well as support for the FORTRAN programming language, and the well-known Vampir MPI trace analyzer). An advantage of the HPC tools on Windows is that all parts are seamlessly integrated with each other, and all parts define a standard for ISVs. The workshop presented the HPC development environment on Windows for programs parallelized in OpenMP, Threading Building Blocks and MPI. It covered the Visual Studio 2008, Microsoft

HPC Server 2008, Allinea DDTlite and the Intel Parallel Studio. The goal was to acquaint the attendee with the tools typically used in a HPC development work-flow: program runtime analysis over parallel debuggers, correctness checking tools, parallel profilers, and the batch system.

6 Solving Larger Problems with MATLAB: A Parallel Computing with MATLAB Hands-On Workshop

Sarah Zaranek and Bryan Zocco, *Mathworks Inc.*

Abstract. The session described how to speed up simulations and to handle larger data sets using parallel computing techniques with MATLAB. The focus was on development and parallel execution of applications on a multi-core desktop computer and on how to scale-up an application to a cluster of computers.

Table of Contents

Turbulence

Direct Numerical Simulation of Fully-Developed Turbulent Channel Flow Using the Lattice Boltzmann Method and Analysis of OpenMP Scalability	1
---	---

Dustin Bespalko, Andrew Pollard, and Mesbah Uddin

Parallel Computations of Unsteady Three-Dimensional Flows in a High Pressure Turbine	20
--	----

Dongil Chang and Stavros Tavoularis

Application of Parallel Processing to the Simulation of Heart Mechanics	30
---	----

Matthew G. Doyle, Stavros Tavoularis, and Yves Bourgault

A Hybrid Parallel Algorithm for Transforming Finite Element Functions from Adaptive to Cartesian Grids	48
--	----

Oliver Fortmeier and H. Martin Bücker

Leveraging Multicore Cluster Nodes by Adding OpenMP to Flow Solvers Parallelized with MPI	62
---	----

Christian Iwainsky, Samuel Sarholz, Dieter an Mey, and Ralph Altenfeld

Materials & Life Sciences

A Maxwell-Schrödinger-Plasma Model and Computing Aspects for Intense, High Frequency and Ultrashort Laser-Gas Interaction	70
---	----

Emmanuel Lorin and André Dieter Bandrauk

The Implementation of Polarizable and Flexible Models in Molecular Dynamics Simulations	76
---	----

Shihao Wang and Natalie M. Cann

Efficient Numerical Methods for the Time-Dependent Schroedinger Equation for Molecules in Intense Laser Fields	99
--	----

André Dieter Bandrauk and HuiZhong Lu

A Parallel Algorithm for Computing the Spectrum of CH_5^+	109
--	-----

Xiao-Gang Wang and Tucker Carrington Jr.

Modeling the Interactions between Poly(N-Vinylpyrrolidone) and Gas Hydrates: Factors Involved in Suppressing and Accelerating Hydrate Growth	117
<i>Brent Wathen, Peter Kwan, Zongchao Jia, and Virginia K. Walker</i>	
Nonlinear Time-Dependent Density Functional Theory Studies of Ionization in CO ₂ and N ₂ by Intense Laser Pulses and Molecular Orbital Reconstruction	134
<i>Emmanuel Penka Fowe and André Dieter Bandrauk</i>	
Considerations for Progressive Damage in Fiber-Reinforced Composite Materials Subject to Fatigue	148
<i>John Montesano, Kamran Behdinan, Zouheir Fawaz, and Cheung Poon</i>	
Bringing HPC to Industry	
Parallelizing Power Systems Simulation for Multi-core Clusters: Design for an SME	165
<i>Hossein Pourreza, Ani Gole, Shaahin Filizadeh, and Peter Graham</i>	
Computing Science, Mathematics, & Statistics	
OpenMP Parallelization of a Mickens Time-Integration Scheme for a Mixed-Culture Biofilm Model and Its Performance on Multi-core and Multi-processor Computers	180
<i>Nasim Muhammad and Hermann J. Eberl</i>	
An Evaluation of Parallel Numerical Hessian Calculations	196
<i>Mark S. Staveley, Raymond A. Poirier, and Sharene D. Bungay</i>	
Preliminary Findings of Visualization of the Interruptible Moment	215
<i>Edward R. Sykes</i>	
SCTP, XTP and TCP as Transport Protocols for High Performance Computing on Multi-cluster Grid Environments	230
<i>Diogo R. Viegas, R.P. Mendonça, Mario A.R. Dantas, and Michael A. Bauer</i>	
Dynamic Resource Matching for Multi-clusters Based on an Ontology-Fuzzy Approach	241
<i>Denise Janson, Alexandre P.C. da Silva, Mario A.R. Dantas, Jinhui Qin, and Michael A. Bauer</i>	
Domain Decomposition of Stochastic PDEs: A Novel Preconditioner and Its Parallel Performance	251
<i>Waad Subber and Abhijit Sarkar</i>	

Interactive Data Mining on a CBEA Cluster.....	269
<i>Sabine McConnell, David Patton, Richard Hurley, Wilfred Blight, and Graeme Young</i>	
HPC Systems & Methods	
Service-Oriented Grid Computing for SAFORAH	283
<i>Ashok Agarwal, Patrick Armstrong, Andre Charbonneau, Hao Chen, Ronald J. Desmarais, Ian Gable, David G. Goodenough, Aimin Guan, Roger Impey, Belaid Moa, Wayne Podaima, and Randall Sobie</i>	
IO Pattern Characterization of HPC Applications	292
<i>Jeffrey Layton</i>	
Failure Data-Driven Selective Node-Level Duplication to Improve MTTF in High Performance Computing Systems.....	304
<i>Nithin Nakka and Alok Choudhary</i>	
Out-of-Core Parallel Frontier Search with MapReduce	323
<i>Alexander Reinefeld and Thorsten Schütt</i>	
Hybrid MPI-Cell Parallelism for Hyperbolic PDE Simulation on a Cell Processor Cluster	337
<i>Scott Rostrup and Hans De Sterck</i>	
<i>BW-DCache: An Inexpensive, Effective and Reliable Cache Solution in a SAN File System</i>	349
<i>Chengxiang Si, Xiaoxuan Meng, and Lu Xu</i>	
Object-Oriented OpenMP Programming with C++ and Fortran	366
<i>Christian Terboven, Dieter an Mey, Paul Kapinos, Christopher Schleiden, and Igor Merkulow</i>	
FFT-Based Dense Polynomial Arithmetic on Multi-cores	378
<i>Marc Moreno Maza and Yuzhen Xie</i>	
Case Study of Scientific Data Processing on a Cloud Using Hadoop.....	400
<i>Chen Zhang, Hans De Sterck, Ashraf Aboulnaga, Haig Djambazian, and Rob Sladek</i>	
Author Index	417

Direct Numerical Simulation of Fully-Developed Turbulent Channel Flow Using the Lattice Boltzmann Method and Analysis of OpenMP Scalability

Dustin Bespalko¹, Andrew Pollard¹, and Mesbah Uddin²

¹ Queen's University, Department of Mechanical and Materials Engineering,
Kingston ON, K7L 3N6, Canada

bespalko@me.queensu.ca, pollard@me.queensu.ca

² The University of North Carolina at Charlotte, Department of Mechanical
Engineering and Engineering Science,
Charlotte NC, 28223-0001 USA
muddin@uncc.edu

Abstract. In this work, the lattice Boltzmann method (LBM) is verified for direct numerical simulation (DNS) of wall-bounded turbulent flows by simulating fully-developed turbulent channel flow and comparing the results to the spectral data of Moser *et al.* [3]. The turbulence statistics compared include: mean velocity and pressure profiles, Reynolds stress profiles, skewness and flatness factors, the turbulence kinetic-energy budget, and one-dimensional energy spectra. Additionally, a scalability test is performed for the implementation of the LBM parallelised with OpenMP for shared-memory architectures. The effect of the domain decomposition algorithm is studied by comparing the performance of a channel flow simulation decomposed with a naïve decomposition method to a case in which the decomposition is computed using the METIS library [4].

Keywords: Turbulence, direct numerical simulation, lattice Boltzmann method, OpenMP, METIS.

1 Introduction

Numerical simulations of turbulent flows are extremely computationally intensive due to the wide range of length and time scales present in turbulent motion. The size of the largest scales, L , is dictated by the geometry of the flow, while the size of the smallest scales, η , is related to the viscosity of the fluid and the energy dissipation rate. The ratio of these scales is a function of the Reynolds number ($Re = ul/\nu$), which is a dimensionless number that represents the ratio of inertial forces to viscous forces in the flow. It can be shown that the ratio of the smallest scales to the largest scales is proportional to $Re^{-3/4}$ [1]. This means that as the Reynolds number increases, the separation between the largest and smallest scales becomes wider, and the computational cost increases. In fact, the

total number of grid points needed for a turbulent simulation scales with $Re^{9/4}$ [1]. To put this limitation in perspective, in order to simulate the flow over the wing of a Boeing 777, for which the Reynolds number is $O(10^9)$, the number of grid points required would be $O(10^{21})$. This example clearly demonstrates the need for high performance computing in the field of direct numerical simulation of turbulence.

The governing equations for a Newtonian fluid are the Navier-Stokes (NS) equations. If the flow can be assumed to be incompressible, the fluid motion can be described by two conservation equations: the continuity equation (conservation of mass):

$$\frac{\partial u_i}{\partial x_i} = 0 , \quad (1)$$

and the Navier-Stokes (NS) equations (conservation of momentum):

$$\rho \frac{\partial u_i}{\partial t} + \rho u_j \frac{\partial u_i}{\partial x_j} = - \frac{\partial p}{\partial x_i} + \mu \frac{\partial^2 u_j}{\partial x_i^2} . \quad (2)$$

Though analytical solutions exist for various forms of these equations for laminar flows, no solution has yet been found for a turbulent flow. For this reason, research in the field of fundamental turbulence has been restricted to experimental investigation and numerical simulation. The advantage of numerical simulation is that the entire flow field can be interrogated in both space and time.

A direct numerical simulation (DNS) of a turbulent flow is a simulation of the NS equations in which all of the turbulent scales are resolved, and the only approximations made are those of a numerical nature [2]. Direct numerical simulations are normally achieved using spectral or finite volume methods. The accuracy of these methods is well documented in the literature. While spectral methods are highly accurate and computationally efficient, they are typically restricted to simple geometries, particularly those that can be implemented with periodic boundaries. Finite-volume methods, on the other hand, are capable of simulating complex flows but are expensive because they must perform the pressure-velocity coupling operation in physical space. This operation can account for up to 90% of the computational time for a typical finite-volume code. In this paper, we consider the lattice Boltzmann method (LBM) as an emerging competitor to these methods. The advantage of the LBM is that it has a computational efficiency comparable to the spectral method (on a per-node basis), and can be used to simulate complex geometries. However, before it can be used to solve complex flows, its accuracy must be verified. As a precursor, we rigorously test the application of the LBM to turbulent channel flow by comparing with the existing data available from the spectral results of Moser *et al.* [3].

Due to the high computational cost of performing direct numerical simulations, they are only practicable with the use of high performance computing (HPC). Therefore, when analysing the suitability of a particular method for DNS, the scalability must be analysed. In this paper, the parallel performance of an LBM code parallelised with OpenMP for shared-memory machines is studied. Scaling results are presented for the case of turbulent channel flow studied

in this paper, and the effect of altering the domain decomposition algorithm is studied by analysing the performance improvement achieved by using the METIS library [4] to partition the computational domain.

The outline of this paper is as follows: the LBM is introduced in section 2; the flow geometry and parameters used for the channel flow simulation are discussed in section 3; the implementation of the LBM is presented in section 4, which includes a study of the parallel performance; and the results of the channel flow simulation are discussed in section 5 and compared to the database of Moser *et al.* [3].

2 The Lattice Boltzmann Method

In the field of fluid mechanics, fluids are most often assumed to be continuous. The state of a fluid at a given point is described by a set of macroscopic field variables such as density, velocity, pressure, and temperature. If the continuum assumption is valid, these macroscopic field variables are insensitive to the exact position and velocity of the individual molecules that make up the fluid. The governing equations for a continuum fluid are the aforementioned Navier-Stokes equations. If the continuum assumption is not made, the fluid can be modelled as a set of discrete particles. This is called the molecular dynamics (MD) approach. In this method, the conservation laws are applied directly to individual particles and the macroscopic motion of the fluid is obtained from the bulk behaviour of the particles. The advantage of the MD method is that the governing equations are simple; however, the computational cost is large since the number of particles in the model is generally on the order of Avogadro's number ($O(10^{23})$). In order to reduce the computational complexity of the MD method, the particles are often replaced by continuous probability density functions. The Boltzmann equation is an example of a method from this category of statistical mechanics.

The Boltzmann equation, (equation 3), is the governing equation for the single particle distribution function $f(\mathbf{x}, \mathbf{v}, t)$.

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f = J(f) . \quad (3)$$

The single particle distribution function represents the expected mass density of particles located at \mathbf{x} , moving with a velocity \mathbf{v} , at time t . The function on the right-hand-side of equation 3 is the collision operator. The original Boltzmann collision operator is extremely complex [5], and is generally replaced by a simpler model for computational work. The most common collision model used in the LBM is the BGK approximation [6]. The BGK operator is a linear relaxation-type model in which the current single particle distribution is moved toward its equilibrium distribution. The rate at which f approaches its equilibrium state is dictated by a single relaxation parameter ω .

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f = \omega(f^{eq} - f) . \quad (4)$$

Once the single particle distribution is obtained from the solution of the Boltzmann equation, the macroscopic fluid behaviour can be obtained by computing various velocity moments of f . Equations 5 - 7 show the velocity moments that correspond to density, momentum, and energy respectively.

$$\rho = \int f d\mathbf{v} , \quad (5)$$

$$\rho \mathbf{u} = \int \mathbf{v} f d\mathbf{v} , \quad (6)$$

$$\rho e = \int \frac{(v^2 - u^2)}{2} f d\mathbf{v} . \quad (7)$$

Calculating the moments in equations 5 - 7 of the Boltzmann equation recovers the macroscopic conservation equations for mass, momentum, and energy [7]. It can be shown through a multi-scale expansion that these conservation equations are equivalent to the Navier-Stokes equations in the macroscopic limit [8]. The expansion parameter used in the multi-scale expansion is the Knudsen number, which is the ratio of the particle mean free path to a suitable lengthscale in the flow. If only first-order terms in the expansion are retained, the Euler equations for inviscid flow are recovered. If terms up to second-order are included, the multi-scale expansion leads to the Navier-Stokes equations.

The Boltzmann equation with the BGK approximation cannot be solved analytically in general, and thus, numerical methods must be used to find approximate solutions. The lattice Boltzmann method is an efficient numerical scheme that is often used to obtain numerical solutions to the Boltzmann equation [9]. There are three components to the LBM: an evolution equation that has been discretised in both physical and velocity space; a discrete equilibrium distribution; and a procedure to exactly calculate the hydrodynamic moments that are needed to recover the Navier-Stokes equations [10].

Starting from equation 4, the evolution equation for the LBM can be derived by applying an appropriate discretisation scheme in both physical and velocity space. In the LBM, velocity space is discretised by restricting the particles modelled by the Boltzmann equation to travel on a symmetric lattice with a discrete set of velocities. The lattice used in this work is called the D3Q19 lattice (named for its 19 velocities in three dimensions), and is shown in figure 11.

When the velocity argument of the single particle distribution is restricted to only 19 discrete values, equation 4 becomes the discrete Boltzmann equation (DBE):

$$\frac{\partial f_i}{\partial t} + \mathbf{v} \cdot \nabla f_i = \omega(f_i^{eq} - f_i) . \quad (8)$$

Equation 8 is called the discrete Boltzmann equation because it is actually a set of equations; one for each discrete velocity used in the model. It should be noted that $f_i(\mathbf{x}, t) = w_i f(\mathbf{x}, \mathbf{v}_i, t)$. The purpose of the weighting factors (w_i) will be explained later in this section. Next, the DBE is discretised in both space and

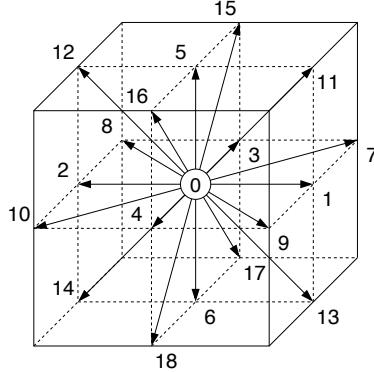


Fig. 1. D3Q19 lattice. Particles are constrained to move with only 19 discrete velocities.

time using a first-order explicit Euler scheme. This yields the lattice Boltzmann equation (LBE), which is the evolution equation for the LBM.

$$f_i(\mathbf{x} + \mathbf{v}_i \Delta t, t + \Delta t) = (1 - \omega) f_i(\mathbf{x}, t) + \omega f_i^{eq}(\mathbf{x}, t) . \quad (9)$$

It should be noted that, despite the use of first-order differences in equation ⑨, the LBM can be made second-order accurate in space and time by using the following expression to link the relaxation parameter to the kinematic viscosity:

$$\nu = \frac{1}{3} \left(\frac{1}{\omega} - \frac{1}{2} \right) . \quad (10)$$

Equation ⑨ can be implemented very efficiently on a uniform cubic mesh by breaking it up into a streaming and collision operation. First, in the collision step, the right-hand-side is evaluated at every grid point and stored as $f'_i(\mathbf{x}, t)$. Then, if the lattice spacing, Δx , is set equal to $v_i \Delta t$, the post-collision particle distributions are streamed to their neighbouring lattice sites by setting $f_i(\mathbf{x} + \mathbf{v}_i \Delta t, t + \Delta t) = f'_i(\mathbf{x}, t)$. This operation is completely explicit and does not require a matrix inversion.

The second component of the LBM is the local equilibrium distribution f_i^{eq} . A logical choice for the local equilibrium distribution is the equilibrium solution of the Boltzmann equation, the Maxwell distribution:

$$f_i^{eq} = w_i \left(\frac{\rho}{(2\pi RT)^{3/2}} \right) \exp \left(\frac{u^2 - v_i^2}{2RT} \right) . \quad (11)$$

In order to simplify this expression, a low-Mach number approximation is used to write the exponential term as an expansion about $\mathbf{u} = 0$. The Mach number is the ratio of the fluid velocity to the speed of sound in the medium.

$$f_i^{eq} = \rho W_i \left[1 + \frac{(\mathbf{v}_i \cdot \mathbf{u})}{RT} + \frac{(\mathbf{v}_i \cdot \mathbf{u})^2}{2(RT)^2} - \frac{u^2}{2RT} \right] , \quad (12)$$

where:

$$W_i = w_i \left(\frac{1}{(2\pi RT)^{3/2}} \right) \exp \left(\frac{-v_i^2}{2RT} \right) . \quad (13)$$

The final component of the LBM is a method for calculating the velocity moments of the discrete single-particle distribution. In the continuous case, the velocity moments were computed by the integrals in equations 5 - 7. However, since velocity space is now discrete, the integrals must be replaced by a quadrature equation that ensures that the velocity moments required to recover the NS equations are exact. In other words, the continuous velocity moments are replaced by the following sums:

$$\rho = \sum_{i=0}^n f_i = \sum_{i=0}^n f_i^{eq} , \quad (14)$$

$$\rho \mathbf{u} = \sum_{i=0}^n \mathbf{v}_i f_i = \sum_{i=0}^n \mathbf{v}_i f_i^{eq} , \quad (15)$$

$$\rho e = \sum_{i=0}^n \frac{(v_i^2 - u^2)}{2} f_i = \sum_{i=0}^n \frac{(v_i^2 - u^2)}{2} f_i^{eq} , \quad (16)$$

where $f_i(\mathbf{x}, t) = w_i f(\mathbf{x}, \mathbf{v}_i, t)$ and $f_i^{eq}(\mathbf{x}, t) = w_i f^{eq}(\mathbf{x}, \mathbf{v}_i, t)$. The quadrature weights, w_i , must be found such that these discrete moments match their equivalent continuous moments. To compute these weights, the velocity moments of the global equilibrium distribution (f^{eq} evaluated at $\mathbf{u} = 0$) are set equal to the exact values of the continuous integrals [10]. The velocity moments up to fourth-order must be exact in order to recover the NS equations. The resulting weight functions give:

$$W_i = \begin{cases} 1/3, & i = 0; \\ 1/18, & i = 1, 2, \dots, 6; \\ 1/36, & i = 7, 8, \dots, 18 \end{cases} \quad (17)$$

Due to the number of degrees of freedom of the local equilibrium distribution, the process of determining the proper weight functions to make the velocity moments exact also fixes the temperature and speed of sound:

$$RT = \frac{v^2}{3} . \quad (18)$$

As a result, the LBM model described in this paper is not capable of solving thermal problems. The pressure at each node can be calculated from the density using the ideal gas law, and since the temperature is fixed this expression becomes:

$$p = \rho \frac{v^2}{3} . \quad (19)$$

Thus, the LBM does not require pressure-velocity coupling to compute the pressure. This represents a significant advantage in terms of computational efficiency since the pressure-velocity coupling can account for up to 90% of the computational time for a finite-volume method.

3 Flow Geometry and Simulation Parameters

In this work, fully developed, plane turbulent channel flow is studied. The purpose of studying this flow is to verify the accuracy of the LBM for a DNS of a wall-bounded turbulent flow. Plane channel flow is an ideal case to study for this purpose due to its simple implementation and the availability of a DNS database for comparison [3].

Plane channel flow involves the flow between two parallel plates with infinite extent in both the streamwise and spanwise directions. Due to the infinite extent, the flow is homogeneous in these directions, which allows for the implementation of periodic boundaries provided that the simulation domain is large enough to contain the largest turbulent structures in the flow. The fluid motion can be driven by either a uniform pressure gradient in the streamwise direction, or an equivalent body force. The latter is preferable in this case since it makes the implementation of the periodic boundaries simpler.

The Reynolds number of the flow, based on the channel half-width (δ), and the mean centreline velocity (U_c), was 3300. This value was chosen to match the spectral simulation of Moser *et al.* [3]. This gives a friction Reynolds number (Re_τ), which is based on the friction velocity u_τ , and the channel half-width, of 180.

The size of the LBM simulation domain was $12\delta \times \delta \times 2\delta$ in the streamwise, spanwise, and wall-normal directions respectively. This domain is considerably smaller in the spanwise direction than the simulation of Moser *et al.* [3], which has dimensions of $4\pi\delta \times 4/3\pi\delta \times 2\delta$. A smaller simulation domain was used in this work to reduce the computational cost of the simulation. Since the LBM must be implemented on a cubic lattice, the grid resolution must be equal in each co-ordinate direction. The computational grid consisted of $1080 \times 90 \times 182$ nodes in the streamwise, spanwise and wall normal directions respectively. This gives a grid resolution of $\Delta x^+ \simeq 2$ (i.e. 2 wall units) in each direction. The resolution of the spectral simulation of Moser *et al.* [3] has a resolution of $\Delta x^+ \simeq 17.7$ in the streamwise direction, and $\Delta y^+ \simeq 5.9$ in the spanwise direction. They used a non-uniform grid in the wall-normal direction with a maximum grid size of $\Delta z^+ \simeq 4.4$ at the centreline of the channel. As a result of the ability to stretch the grid in the streamwise and spanwise directions, Moser *et al.* [3] were able to simulate a larger domain with considerably less computational cost.

The consequence of using a simulation domain that is too short in the spanwise direction is that the channel is not large enough to contain the largest structures in the flow. However, Jiménez *et al.* [11] showed that as long as the dimensions of the channel are sufficiently large to sustain a turbulent flow, the turbulence statistics are reasonably accurate, especially near the wall where the flow is dominated by smaller structures. For a friction Reynolds number of 180, Jiménez *et al.* found that the minimum dimensions for the channel are $\pi\delta \times 0.3\pi\delta \times 2\delta$ in the streamwise, spanwise, and wall-normal directions respectively. Thus, the spanwise extent used in this work is approximately equal to the dimensions of the minimal flow unit. Furthermore, the channel simulated by

Lammers *et al.* [12] also had the same extent in the spanwise direction, and they found good agreement with the database of Moser *et al.* [3].

The turbulence statistics and velocity spectra presented in section 5 were collected once the simulation reached a statistically stationary state. The simulation was first run for approximately 130 large-eddy turnover times (LETOTs) with a coarse resolution ($\Delta x^+ \approx 4$) simulation. A LETOT is defined as the time required for the largest eddies in the simulation to transfer their energy to smaller scales, and is defined as $t u_\tau / \delta$. Once the simulation reached a statistically stationary state, the simulation was run for an additional 70 LETOTs in order to converge the turbulence statistics. The data used to compute the one-dimensional velocity spectra was gathered over the last 15 LETOTs.

4 Implementation and Performance

The D3Q19 LBM was implemented in Fortran 90. Since the code is intended for simulating complex flows, it was implemented in such a way as to be able to simulate flows with complex geometries. For this reason the single particle distributions were stored in unstructured arrays, and a geometry array was used to store the node connectivity. While this data structure is inefficient for simulations with simple geometries, such as plane channel flow, it makes the code adaptable for simulations with more complex geometries.

It should be noted that the actual implementation of the LBM used in this work is the multiple-relaxation time (MRT) LBM [13]. The MRT method transforms the 19 particle distributions, for the D3Q19 lattice, into 19 orthogonal moments. Since these moments are orthogonal, the collision process can be applied directly to the moments, and each moment can be relaxed with a different relaxation time. If all of the relaxation times are set equal, the BGK LBM is recovered. The main advantage of the MRT LBM is that the individual relaxation parameters can be tuned to produce optimal stability for simulations at high Reynolds number. In this work, the individual relaxation times were not tuned since no stability issues were encountered for the channel flow simulations.

The code was parallelised using the OpenMP compiler directives, which allows for simple implementation on shared-memory machines. The simplest approach to parallelise an algorithm with OpenMP is to use loop-level parallelism. However, it was found that this method produced poor scaling due to the large number of fork/join operations required for each iteration of the LBM. Instead, a parallel section was used to implement a domain decomposition approach.

Domain decomposition is most commonly used for parallelising algorithms designed for use on distributed-memory machines. It is well suited to this type of architecture because the data must be divided amongst the memory belonging to each processor. Since each processor is only able to see the data contained in its own memory, a message passing API, such as MPI, must be used to transfer data between processors. On a shared-memory machine, each processor has access to the entire main memory, and thus message passing is not necessary.

The parallel performance of codes parallelised using MPI for distributed-memory systems is mainly determined by the ratio of the communication time to the computation time. If the time required to send messages between processors can be minimised, the parallel performance is maximised. For this reason, performing the domain decomposition efficiently has a significant effect on the scalability. A good domain decomposition is one that evenly distributes the workload across the available processors while minimising the surface area of domain boundaries (edgecut), in order to minimise the size of the messages that must be sent between processors. In simple geometries, determining an efficient domain decomposition is generally straightforward; however, this process becomes more complicated when simulating flows in complex geometries.

A popular package available for performing domain decomposition is the METIS library [4]. METIS converts the computational mesh into a graph, then uses graph partitioning techniques to split the graph into subdomains. METIS uses multilevel graph partitioning techniques that first partition a coarser graph. After the coarse partitioning is complete, the graph is uncoarsened to obtain the partition for the full graph. This technique leads to better quality partitions and higher computational efficiency [4].

Recently, Axner *et al.* [14] applied the METIS library to perform the domain decomposition for their implementation of the LBM. Their code was implemented using MPI for distributed-memory machines, and they found that the METIS decomposition improved the scalability of their code since it minimised load imbalance and edgecut simultaneously. Since our implementation of the LBM has been parallelised using OpenMP, minimising the edgecut is not important from a message passing perspective; however, the METIS decomposition leads to domains with small aspect ratios, which should lead to improved data locality for the particle streaming operation.

To test the performance of the METIS library with the OpenMP implementation of the LBM, a scaling test was performed for the channel flow problem described in section 3. The problem was run with up to 256 threads on a Sun Enterprise M9000 SMP machine at the High Performance Computing Virtual Laboratory (HPCVL). The performance of the case decomposed with METIS was compared to a case in which the domain decomposition was performed by naïvely performing a linear domain decomposition of the nodes. Due to the node numbering, the linear method is roughly equivalent to a one-dimensional domain decomposition in the wall-normal direction. Figure 2 shows a plot of the time required per time-step for both cases. As this plot shows, the performance difference between these cases is negligible.

Figure 3 shows the scaling results for both cases. This plot shows nearly linear scaling out to 32 threads. After this point the scaling is no longer linear; however, a performance gain is still observed from 128 to 256 threads. By extrapolation, it would seem that no performance gains should be expected past 512 threads.

The fact that no performance improvement was observed by using METIS to perform the domain decomposition suggests that the domain decomposition

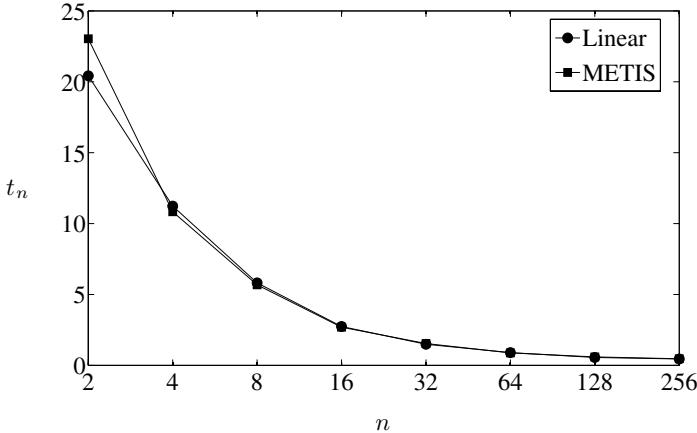


Fig. 2. Computational time for individual time steps for the LBM

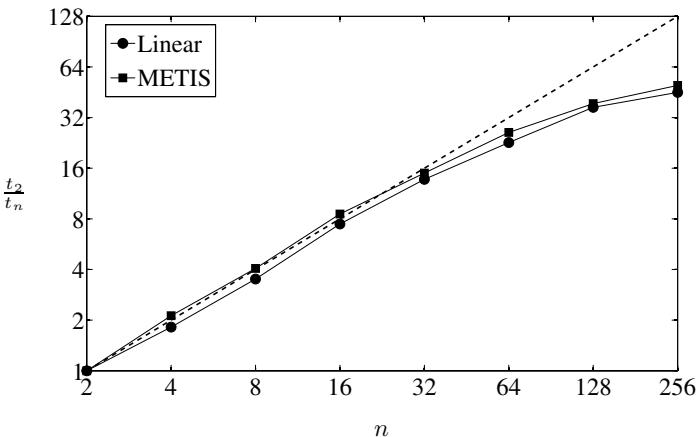


Fig. 3. Strong scaling for the LBM

is not an important factor in the scaling of the LBM for shared-memory environments. While it was hoped that the subdomains created by METIS would lead to better data locality, and thus faster execution times, this was shown not to be the case. The most important factor that affects the scaling performance of the LBM is the ratio of the computational time to the time required for synchronisation. During each time-step, the code must wait after the streaming step for all of the threads to synchronise. If the number of nodes assigned to each thread is large, this synchronisation time is insignificant. However, if a large number of threads are used, this time becomes more important.

The LBM is very computationally efficient on a per time-step basis, but the time-step size for the LBM is small in comparison to more traditional CFD

methods. For this reason, the LBM does not scale well for small problems. However, as figure 3 shows, the LBM scales well for large problems, such as direct numerical simulations.

5 Results

In this section, the results from the LBM channel flow simulation described in section 4 are presented and compared to the spectral results of Moser *et al.* [3]. Figure 4 shows a plot of the mean velocity profile normalised by the friction velocity. The LBM simulation shows a good agreement with the spectral results; however, the slope of the velocity profile is slightly steeper by approximately 8% in the log-law region. This is most likely due to the small spanwise extent of the channel since a coarse resolution ($\Delta x^+ \simeq 4$) LBM simulation with a larger spanwise extent (run previously), showed a better agreement with the slope of the database in the log-law region. There also appears to be a wake region at the centre of the channel for the LBM simulation. This seems to also be caused by the small spanwise extent since a wake region was not observed in the coarse simulation.

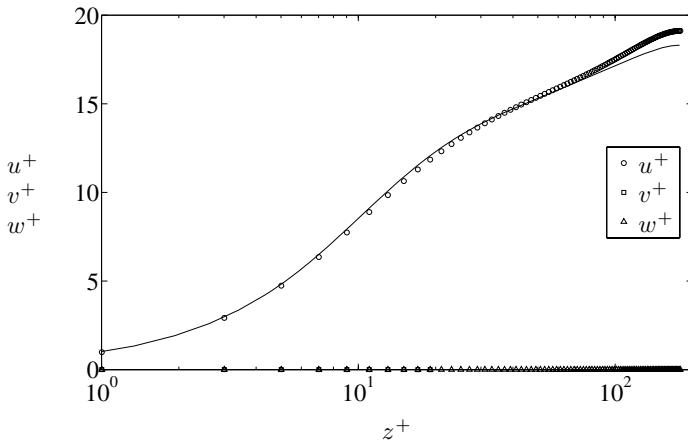


Fig. 4. Mean velocity profiles for the LBM simulation (points) and Moser *et al.* [3] (lines)

A plot of the mean pressure profile is shown in figure 5. This plot also shows good agreement between the LBM simulation and the spectral results of Moser *et al.* [3]. As with the mean velocity profile, there is a small discrepancy in the log-law region, which is again most likely a result of the narrow simulation domain. The mismatch at the centreline is also due to the limited spanwise extent, as was the appearance of a wake region in the mean velocity profile.

Figure 6 shows the profiles for the Reynolds normal stresses for the LBM simulation, and the Reynolds shear stresses are plotted in figure 7. In general,

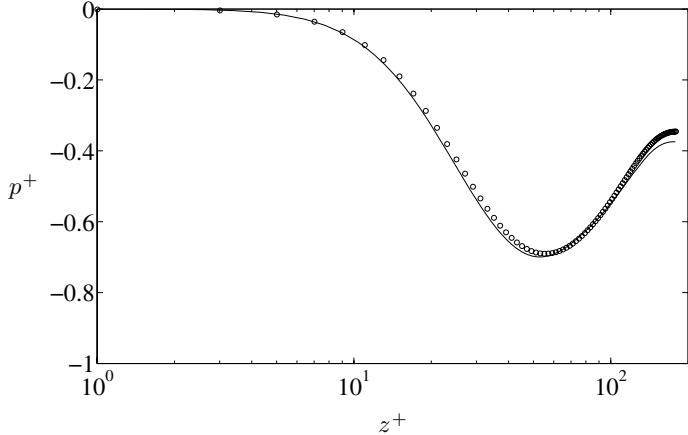


Fig. 5. Mean pressure profile for the LBM simulation (points) and Moser et al. [3] (lines)

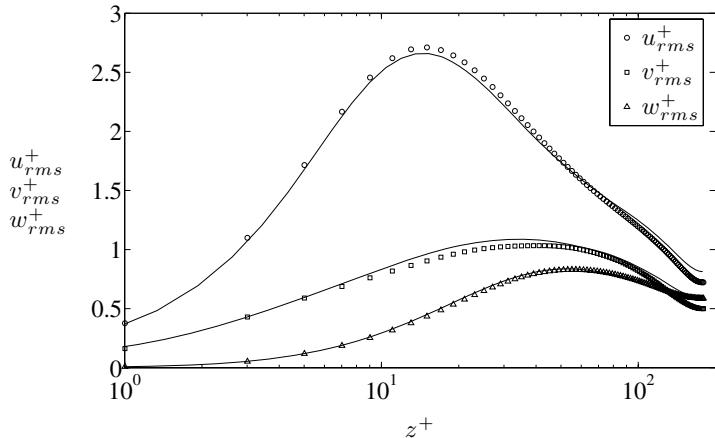


Fig. 6. Reynolds normal stress profiles for the LBM simulation (points) and Moser et al. [3] (lines)

the results from the LBM simulation match the curves from the database reasonably well. However, the streamwise Reynolds stress is over-predicted by approximately 3% in the buffer layer, and the spanwise Reynolds stress is similarly under-predicted. This discrepancy is also due to the narrow extent in the spanwise direction. Previous simulations performed with the LBM showed that the streamwise and spanwise Reynolds stresses in the buffer region are sensitive to the size of the computational domain. An overshoot of approximately 10% was observed for the streamwise Reynolds stress when the streamwise extent of the channel was equal to $\pi\delta$. Additionally, the aforementioned coarse resolution

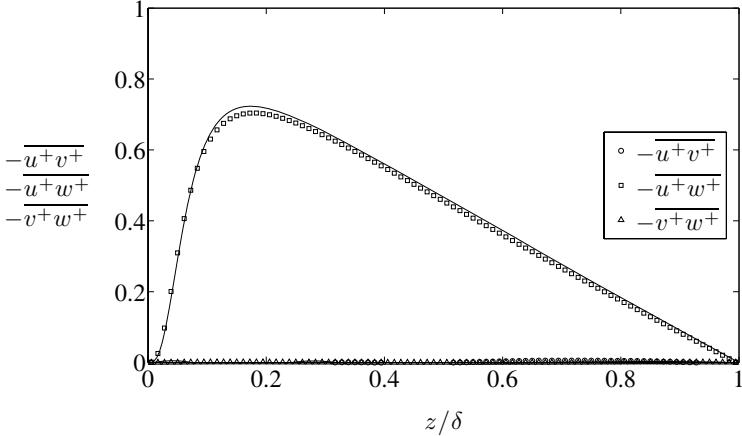


Fig. 7. Reynolds shear stress profiles for the LBM simulation (points) and Moser *et al.* [3] (lines)

simulation with a wider spanwise extent showed a negligible overshoot for the streamwise Reynolds stress, and no under-prediction of the spanwise Reynolds stress profile.

Another discrepancy between the LBM solution and the database is that the peak Reynolds shear stress predicted by the LBM is 3% lower than the one from the database. Since the coarse LBM simulation under-predicts the peak Reynolds shear stress by 8%, this suggests that this problem is a result of insufficient grid resolution in the buffer region. As mentioned in section 3, the resolution of the LBM simulation was chosen to match the Kolmogorov length scale in the channel as reported by Kim *et al.* [15]. A higher resolution in the buffer layer would have been used; however, since the LBM is implemented on a uniform cubic lattice, an increase in the resolution in the wall-normal direction must be applied in all co-ordinate directions. Thus, a lower resolution was chosen in order to reduce the computational time required for the simulation. It is expected that increasing the resolution further would improve the match.

Figure 8 shows a plot of the viscous shear stresses. The viscous shear stress and the Reynolds shear stress add together to offset the action of the applied body force. Since the body force is constant, the viscous and Reynolds shear stresses add to create a total shear stress profile that is linear. As a result of the under-prediction of the Reynolds shear stress, a corresponding over-prediction occurs in the plot of the viscous shear stresses. Again, increasing the grid resolution is expected to remove this discrepancy.

The skewness and flatness factors for the velocity field were calculated for the LBM simulation and compared to the spectral database. The profiles for the LBM simulation and the spectral results of Moser *et al.* [3] are plotted in figures 9 and 10. Unfortunately the spectral results are not properly converged for these statistics. Higher-order statistics require larger integration times to reach convergence, and the simulation of Moser *et al.* [3] was not integrated

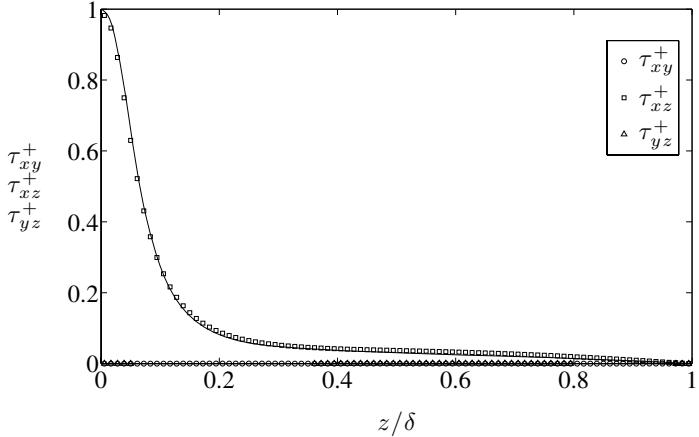


Fig. 8. Viscous shear stress profiles for the LBM simulation (points) and Moser *et al.* [3] (lines)

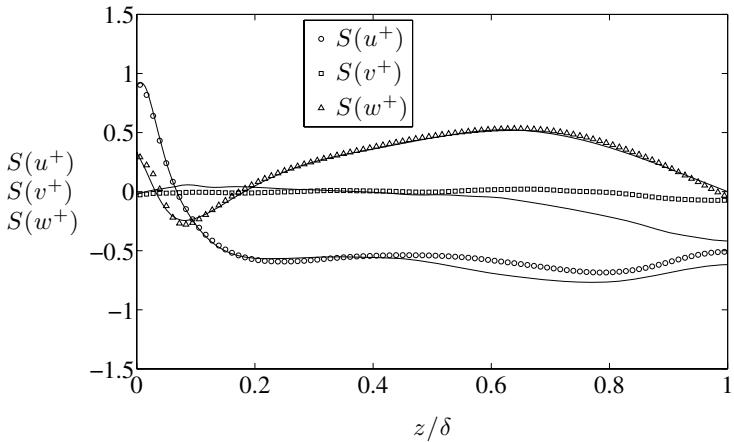


Fig. 9. Skewness for the LBM simulation (points) and Moser *et al.* [3] (lines)

long enough in order for the skewness and flatness factors to converge. This is indicated by the presence of oscillations in the profiles, and the fact that the skewness of the wall-normal velocity fluctuations is not zero [15]. Since the length and time scales are smaller close to the wall than near the centre of the channel, the turbulence statistics tend to converge faster near the wall. Thus, it is expected that the LBM simulation should match the database results near the wall. As figures 9 and 10 show, the LBM simulation is in good agreement with the spectral results for $z/\delta < 0.5$.

The turbulence kinetic-energy budget for the LBM simulation and the spectral simulation are plotted in figure 11. As this plot shows, the LBM simulation is in excellent agreement with the database. The only notable discrepancy between

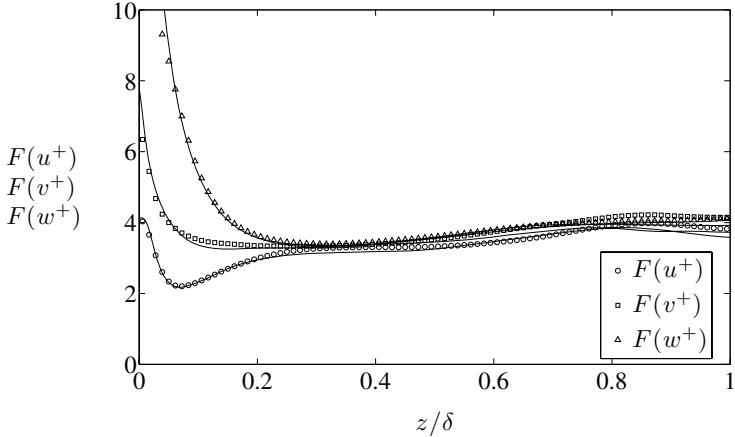


Fig. 10. Flatness for the LBM simulation (points) and Moser *et al.* [3] (lines)

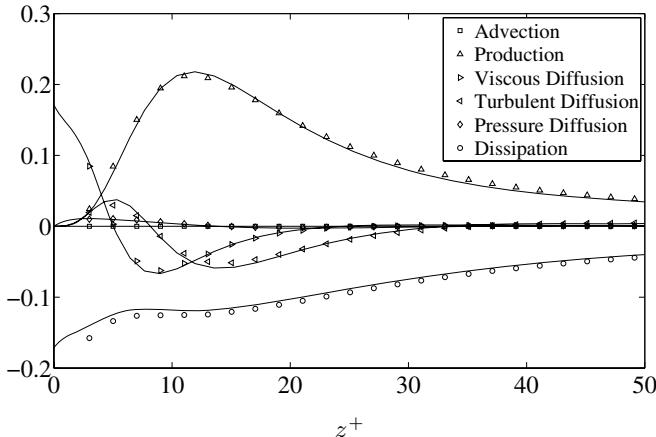


Fig. 11. Turbulence kinetic-energy budget for the LBM simulation (points) and Moser *et al.* [3] (lines)

the two simulations is that the LBM predicts higher viscous dissipation near the wall. This effect is most likely due to insufficient grid resolution near the wall. The turbulence kinetic-energy budget, especially the viscous dissipation term, was found to be very sensitive to the grid resolution. The coarse simulation with a resolution of $\Delta x^+ \simeq 4$ predicted a much higher value for the viscous dissipation near the wall. Increasing the resolution is expected to improve the match with the database simulation.

The one-dimensional energy spectra were computed for the LBM simulation and are compared to the database in figures 12 and 13. Figure 12 shows the streamwise one-dimensional energy spectra for each component of the velocity

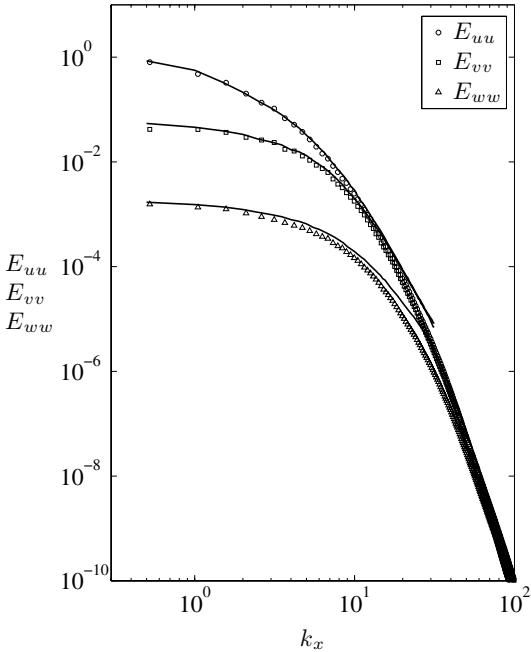


Fig. 12. Streamwise one-dimensional energy spectra at $z^+ \simeq 5$ for the LBM simulation (points) and Moser *et al.* [3] (lines)

fluctuations. The spectra plotted are computed at a distance of $z^+ \simeq 5$ from the wall. The spectra show an excellent agreement with the results of Moser *et al.* [3], particularly in the low-wavenumber range. The LBM spectra deviate slightly from the database results for the highest wavenumbers contained in the spectral simulation. This deviation suggests that the resolution of the spectral simulation in the streamwise direction was not quite sufficient to resolve the smallest scales, and thus a slight energy pile-up is observed in the spectra. However, since the energy density at the highest wavenumbers is several orders of magnitude lower than the energy density at the low-wavenumbers, it's unlikely that this pile-up significantly affects the turbulence statistics. This plot demonstrates the fact that the LBM simulation is significantly over-resolved in the streamwise direction. The LBM energy spectra show more than 10 decades difference between the energy density at the high-wavenumbers and the energy density at the low-wavenumbers.

Figure 13 shows that the spanwise one-dimensional energy spectra for the LBM simulation are also in good agreement with the spectral database. The spanwise LBM spectra show a slight deviation at high-wavenumber like the streamwise spectra, but this deviation is smaller since the spanwise resolution for the spectral simulation was nearly three times finer than the streamwise direction.

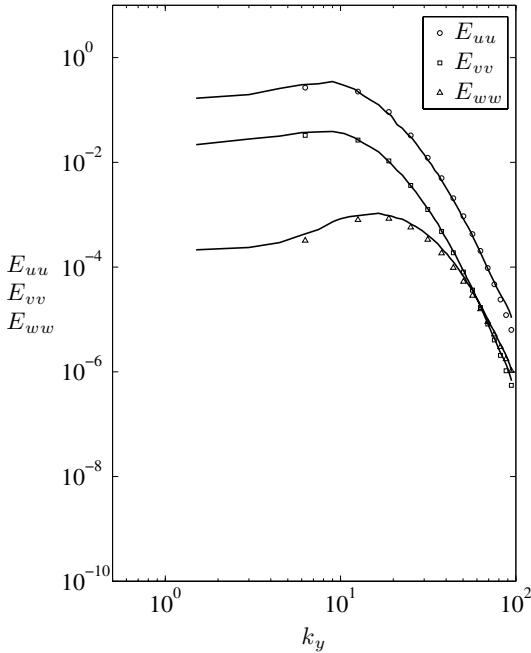


Fig. 13. Spanwise one-dimensional energy spectra at $z^+ \simeq 5$ for the LBM simulation (points) and Moser *et al.* [3] (lines)

The energy spectra near the centre of the channel were computed for the LBM simulation as well, and were also compared to the database. These spectra showed similar agreement with the spectral results, and are not shown in this paper.

6 Conclusions

In this work the LBM was verified for DNS of wall-bounded turbulent flows by simulating fully-developed turbulent channel flow and comparing the turbulence statistics, turbulence kinetic-energy budget, and one-dimensional energy spectra to the spectral results of Moser *et al.* [3]. Additionally, the scalability of the LBM code was analysed for shared-memory machines, and the effect of varying the domain decomposition was investigated. The specific conclusions from this work are:

- In general, the LBM simulation showed good agreement with the spectral results of Moser *et al.* [3]. However, there were a number of discrepancies between the two datasets caused by the relatively coarse resolution of the LBM simulation in the wall-normal direction, and the reduced extent of the computational domain in the spanwise direction. The coarse resolution of the LBM simulation in the wall-normal direction resulted in an under-prediction of the

Reynolds shear stress in the buffer layer, and an over-prediction of the viscous shear stresses. The small extent in the spanwise direction caused the slope of the mean velocity profile to become steeper in the log-law region, and also lead to discrepancies in the Reynolds normal stress profiles.

- The turbulence kinetic-energy budget from the LBM simulation matched the database results except for an over-prediction of the viscous dissipation term near the wall. Coarse resolution simulations showed that the turbulence kinetic-energy budget is very sensitive to the grid resolution, so it is expected that this match would improve for a simulation with a higher resolution in the wall-normal direction.
- The one-dimensional energy spectra were in good agreement for the LBM simulation and the spectral results. The LBM spectra deviated slightly from the database results in the high-wavenumber range. This effect can most likely be attributed to the much higher resolution of the LBM simulation compared to the database in the streamwise and spanwise directions. The relatively coarse resolution of the spectral simulation in the homogeneous directions causes a slight energy pile-up at the highest wavenumbers included in the model. Since the resolution in the streamwise and spanwise directions was much higher for the LBM simulation it does not exhibit this energy pile-up.
- The LBM algorithm was found to scale linearly up to 32 threads on the Sun Enterprise M9000 Server SMP machine. Performance gains were observed up to 256 threads, and it is estimated that the peak performance is achieved with approximately 512 threads. The scaling tests were performed for the channel flow problem described in this paper, which contains approximately 17.7 million nodes. Better scaling is expected for larger problems.
- The method of performing domain decomposition was found to have a negligible effect on the parallel performance of the LBM for SMP machines. The performance of the LBM code decomposed with a linear domain decomposition algorithm was compared to a case in which the decomposition was performed using the METIS library [4]. While the METIS library created subdomains with smaller aspect ratios and less surface area, the computational performance improvement was negligible. Thus, the performance of LBM codes implemented in OpenMP for SMP machines was not found to be sensitive to changes in the domain decomposition strategy.
- While the computational efficiency of the LBM is comparable to the spectral method on a per node basis, the computational cost of simulating channel flow with the LBM is significantly larger due to the use of the uniform cubic lattice. The spectral method can be implemented with different grid resolutions in the three co-ordinate directions. For this reason the grid resolution can be increased in the wall-normal direction without increasing the resolution in the streamwise and spanwise direction. This gives the spectral method, or even a finite volume method, a significant advantage over the LBM for simple geometries. However, as the geometry of the computational domain becomes increasingly complex, this advantage should become less significant.

Acknowledgements

The authors would like to thank the Natural Science and Engineering Research Council (NSERC) postgraduate scholarship program for financial support, and the Sun Microsystems of Canada Scholarships in Computational Science and Engineering. Computational resources were provided by the High Performance Computing Virtual Laboratory (HPCVL) and by Cray Inc.

References

1. Pope, S.B.: *Turbulent Flows*. Cambridge University Press, Cambridge (2000)
2. Guerts, B.J.: *Elements of Direct and Large-eddy Simulation*. Edwards (2004)
3. Moser, R.D., Kim, J., Mansour, N.N.: Direct numerical simulation of turbulent channel flow up to $Re-\tau = 590$. *Phys. Fluids* 11, 943–945 (1999)
4. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20, 359–392 (1998)
5. Boltzmann, L.: *Lectures on Gas Theory*. University of California Press (1964)
6. Bhatnagar, P.L., Gross, E.P., Krook, M.: A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems. *Phys. Rev.* 94, 511–525 (1954)
7. Harris, S.: *An Introduction to the Theory of the Boltzmann Equation*. Dover Publications Inc. (2004)
8. Chapman, S., Cowling, T.: *The Mathematical Theory of Non-Uniform Gases: An Account of the Kinetic Theory of Viscosity, Thermal Conduction and Diffusion in Gases*. Cambridge University Press, Cambridge (1970)
9. He, X.Y., Luo, L.S.: Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation. *Phys. Rev. E* 56(6), 6811–6817 (1997)
10. Wolf-Gladrow, D.A.: *Lattice-Gas Cellular Automata and Lattice Boltzmann Models*. Springer, Heidelberg (2000)
11. Jiménez, J., Moin, P.: The minimal flow unit in near-wall turbulence. *J. Fluid Mech.* 225, 213–240 (1991)
12. Lammers, P., Beronov, K.N., Volkert, R., Brenner, G., Durst, F.: Lattice BGK direct numerical simulation of fully developed turbulence in incompressible plane channel flow. *Comput. Fluids* 35, 1137–1153 (2006)
13. Lallemand, P., Luo, L.S.: Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability. *Phys. Rev. E* 61, 6546–6562 (2000)
14. Axner, L., Bernsdorf, J., Zeiser, T., Lammers, P., Linxweiler, J., Hoekstra, A.G.: Performance evaluation of a parallel sparse lattice Boltzmann solver. *J. Comput. Phys.* 227, 4895–4911 (2008)
15. Kim, J., Moin, P., Moser, R.: Turbulence statistics in fully-developed channel flow at low Reynolds-number. *J. Fluid Mech.* 177, 133–166 (1987)

Parallel Computations of Unsteady Three-Dimensional Flows in a High Pressure Turbine

Dongil Chang and Stavros Tavoularis

Department of Mechanical Engineering, University of Ottawa,
Ottawa, ON Canada
Stavros.Tavoularis@uottawa.ca

Abstract. Steady (RANS) and unsteady (URANS) numerical simulations of three-dimensional transonic flow in a high-pressure turbine have been conducted. The results are compared by evaluating differences in contours of total pressure loss, contours of static pressure and wakes. Significant differences have been observed in these parameters, both qualitative and quantitative, which demonstrate that URANS simulations are necessary when the effect of stator-rotor interaction on flows in turbines is strong.

Keywords: Turbine, coherent vortices, blade, vane, RANS, URANS.

1 Introduction

Turbomachines are used extensively in transportation (e.g., aircraft engines and propellers), energy generation (e.g., steam turbines and wind turbines), and fluid moving (e.g., pumps, fans and blowers). They span, in size, the range from micrometers for a micro-turbine to 150 m for a wind turbine, and they are available in a great variety of shapes. Industrial design of turbomachines relies heavily on semi-empirical design tools, notably on correlations among various parameters, which are based on experimental results, sometimes collected under idealized conditions. The accuracy of such approaches faces serious challenges and their applicability is limited within their range, which reduces their usefulness for new designs. With advances in computational resources and three-dimensional simulation tools, large-scale steady CFD (Computational Fluid Dynamics) simulations are becoming more common in the design process of turbomachines.

As a result of the continuing efforts to produce lighter and smaller gas turbines for the aviation industry, the turbine stage loading is increased, while the space between blade rows is reduced. This leads to increased effects of unsteadiness and three-dimensionality. Unsteady three-dimensional effects are particularly strong in flow fields through transonic high-pressure turbines, in which they have been associated with the presence of coherent vortices, wakes, and trailing-edge shock waves. The vibratory stresses on a turbine blade, which are intimately connected with the lifetime of the engine and the scheduling of maintenance inspections, are highly dependent on the flow unsteadiness, due to the stator-rotor interactions. This interaction is also known as forced response and is more important in a high-pressure turbine than in a

low-pressure turbine. To determine the vibratory stress level on turbine blade surfaces, one requires the time-dependent surface pressure distribution on the blades. Experimental determination of this distribution is extremely difficult, which makes unsteady CFD simulations a valuable tool for high-pressure turbine performance analysis and design.

In recent years, advanced unsteady CFD methods, including Direct Numerical Simulations (DNS) and Large Eddy Simulations (LES), have been introduced in the study of turbomachinery flows in low-pressure turbines [1-3] and in high-pressure turbine cascades [4]. Nevertheless, their application to the simulation of flows in high-pressure turbines has not yet been accomplished, because they require excessive computer resources. A less powerful but more practical approach, which is feasible with the use of currently available computer systems, is the numerical solution of the Unsteady Reynolds-Averaged Navier-Stokes (URANS) equations, with the inclusion of a suitable turbulence model. The computational resources and CPU time required for URANS simulations are much larger than those required for steady (RANS) ones, and it is necessary to investigate whether the benefits from the use of the former are worth the additional efforts and cost.

In the present article, which follows a previous one [5], we will compare time-averaged results of URANS simulations with corresponding results of RANS simulations in order to assess the significance of unsteadiness on turbine analyses.

2 Computational Procedures

The computational geometry was the same as that in our previous simulations [5]. Simulations were conducted using the commercial code FLUENT 6.3. Considering the compressible nature of the flow, the implicit coupled solver was used by solving the momentum and energy equations at the same time. For high accuracy, a second-order upwind scheme was chosen for space discretization and the second-order implicit Euler scheme was employed for temporal discretization.

The simulations are the solutions of the URANS equations with the SST turbulence model, which is a combination of the $k-\omega$ model in the near-wall region and the $k-\varepsilon$ model in the core flow region [6]. Rotational periodic boundary conditions were employed, based on the assumption that, like the geometry, the velocity distribution was also azimuthally periodic. No-slip conditions were applied at all walls. The adiabatic boundary condition was applied on all walls, following common practice in turbomachinery analyses [7]. The total pressure and the total temperature were specified at the inlet, while the static pressure was specified at the outlet assuming radial equilibrium. The inlet Mach number was approximately 0.1.

Figure 1 shows the computational domain, which consisted of an inlet section containing a stator with one vane passage, a rotating rotor section having three blade passages, and a non-rotating outlet duct section. The inlet plane was located in the middle of the 180 degree duct, which connected the combustor with the high-pressure transonic turbine stage. The outlet duct section had a length of two blade axial chords and was included to eliminate any possible interactions of the flow in the rotor with reflected pressure waves generated on the blades at the turbine outlet.

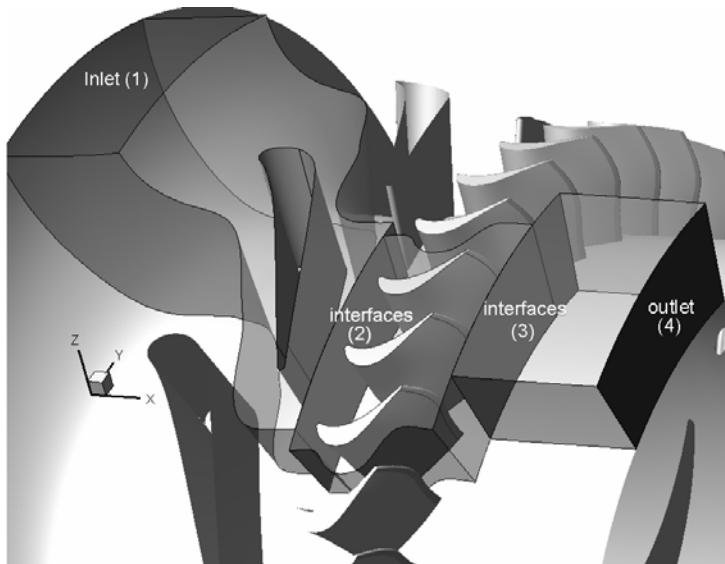


Fig. 1. Computational geometry: (1) inlet; (2) sliding interface between the stator and the rotor; (3) sliding interface between the rotor and the extended duct; (4) outlet

RANS simulations were performed using the mixing plane implementation and steady solutions served as the initial input for URANS simulations. In the URANS simulations, the rotor domain was given a constant angular velocity and sliding meshes were used to model the interface regions. Instantaneous solutions were advanced in time using a dual time stepping technique, which accounts for the relative motion of rotor and stator and other sources of unsteadiness.

The initial time step was set sufficiently small for the solution to converge within 20 iterations for every time step. It was increased subsequently to reduce the overall computing time, but the increase was gradual to ensure convergence within each time step. 200,000 time steps per rotor blade passing were used initially, subsequently reduced to 500, which were small enough to resolve time-varying flow features.

The convergence of solution was checked by monitoring the temporal fluctuations of the mass flow rate at the outlet, shown in Fig. 2(a). Inspection of this figure demonstrated that a time-periodic mass flow rate was achieved at $6t/\tau_v$, where τ_v is the vane passing period (i.e., the time it takes for one rotor blade to pass past two adjacent stator vanes). Convergence levels of the solution's periodicity were also determined by following the suggestion of Clark and Grover [8]. Figure 2(b) shows plots of convergence levels for the mean mass flow rate, cross-correlation factor (CCF) at zero lag, and fractional power of the power spectrum, as well as the overall convergence level (OCE; i.e., the minimum value of the previous convergence levels). All examined convergence levels at $6t/\tau_v$ reached values which were much closer to 1 (perfect periodicity) than the convergence threshold (CT) of 0.950, which Clark and Grover [8] suggested to be sufficient for overall convergence.

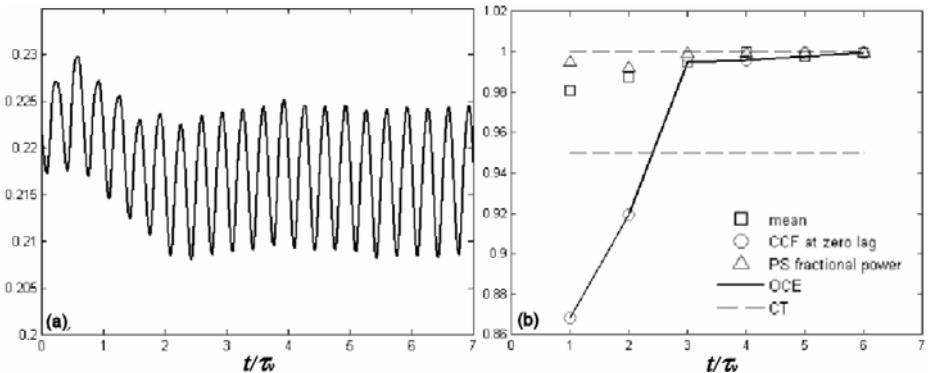


Fig. 2. Time evolution (a) and convergence levels (b) of the mass flow rate at the outlet

An unstructured mesh was used, which composed of tetrahedral elements in the core region and prismatic hexahedral elements in the boundary regions. 10 prism mesh elements were allocated across the gap regions between the blade tip and the rotor casing.

Three different meshes (1.2, 1.7 and 2.5 Million cells) were used to examine the sensitivity of mesh size on the numerical accuracy and the medium mesh with 1.7 Million cells was selected as combining a high accuracy and a relatively low computing time [5].

The parallel-computing performance of FLUENT 6.3 with MPI was examined in a Sunfire cluster at HPCVL (High Performance Computing Virtual Laboratory), configured with four Sun Fire 25000 servers with 72 dual UltraSPARC-IV+ processors running at 1.5 GHz. Because the maximum allowable thread number for each user of HPCVL is limited to 20, the selected test cases had numbers of threads N within the range between 5 and 20. Figure 3 shows the parallel performance of FLUENT in a Sunfire cluster in terms of speedup and efficiency. The speedup is defined as the ratio of the wall clock time for $N = 5$ and the wall clock time for $N > 5$ (it is noted that the usual definition of speedup is with respect to one thread, but single-thread simulations were not performed in this work), whereas the efficiency is defined as the ratio of the speedup and N. When $N=5$, it took 30.4 minutes of wall clock time for one time step. Figure 3(a) indicates a very good speedup performance with increasing N within this range. The parallelization efficiency in Fig. 3(b) indicates that the efficiency did not change substantially and that it reached a maximum for $N=15$.

The same job that was run on the Sunfire cluster was also run on a cluster at the University of Ottawa, configured with six Dell HP M8120N servers with 12 quad core Intel 6600 processors running at 2.4 GHz. Simulations on the Dell cluster with 20 threads were about 1.7 times faster than those on the Sunfire cluster. Subsequently, all main computations were run on the Dell cluster. It took about 9 days to collect data for one vane passing period.

The steady RANS simulations were run on a personal computer with 2 quad core Intel 6600 processors at 2.4 GHz. It took 23 hours for about 4000 iterations, which were required for the convergence of the solutions.

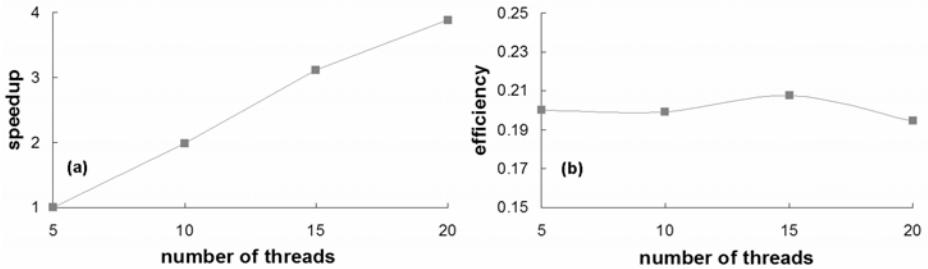


Fig. 3. Dependence of speedup (a) and efficiency (b) on the number of threads used for the Sunfire cluster

3 Results

The modified Q criterion, introduced by Chang and Tavoularis [9], allowed us to identify all well-known types of quasi-stationary vortices in axial gas turbines, including the stator casing horseshoe and passage vortices, the rotor casing and hub horseshoe and passage vortices, the blade tip leakage vortices and vortices shed by the trailing edges of vanes and blades. The modified Q is defined as

$$Q_m = \frac{1}{2} (\Omega_{ij} \Omega_{ij} - c_q S_{ij} S_{ij}) \quad (1)$$

where $\Omega_{ij} = \frac{1}{2} \left(\frac{\partial U_i}{\partial x_j} - \frac{\partial U_j}{\partial x_i} \right)$ and $S_{ij} = \frac{1}{2} \left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)$ are the rotation tensor and the rate of strain tensor, respectively, and c_q is an empirical factor.

In the original Q criterion, $c_q = 1$ and positive values of Q identify vortices. However, the identification of vortices by the original Q criterion is highly dependent on the choice of the Q value, called the threshold. Using values of $c_q \neq 1$ increases or decreases the relative effect of the rate of strain and may identify realistic coherent vortices, which are less sensitive to the threshold choice [9]. In the present work, the value $c_q = 0.63$ and the threshold value of 1 s^{-2} were chosen as appropriate.

All these vortices in the rotor section have been marked in Fig. 4. In addition to these quasi-stationary vortices, Chang and Tavoularis [5], using URANS, documented two types of unsteady vortices, the axial gap vortices and the crown vortices. The formation of both types is attributed to interactions between vane wakes and passing blades. The axial gap vortices are roughly aligned with the leading edges of the rotor blades, whereas the crown vortices are located in the mid-section of the blade passage near the blade crown. These vortices have also been marked in Fig. 4.

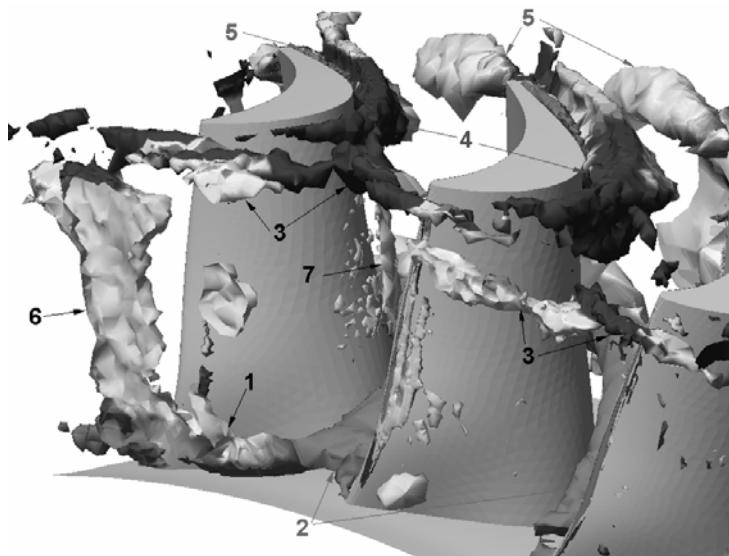


Fig. 4. Vortices in the rotor (coloured by axial vorticity): hub horseshoe (1), hub passage (2), casing horseshoe (3), casing passage (4), tip leakage (5), axial gap (6), crown (7)

Coherent vortices identified in the rotor using RANS and URANS are presented in Fig. 5. The sizes and the locations of the rotor casing passage vortices and the rotor hub passage vortices are comparable in both simulations. However, the horseshoe vortices and the tip leakage vortices in RANS are smaller than those in URANS; the tip leakage vortices in URANS start rolling up near the mid-chord section, while the tip leakage vortices in RANS first appear half way between the mid-chords and the trailing edges of blades. Finally, the axial gap vortex observed in URANS is not at all identified in RANS.

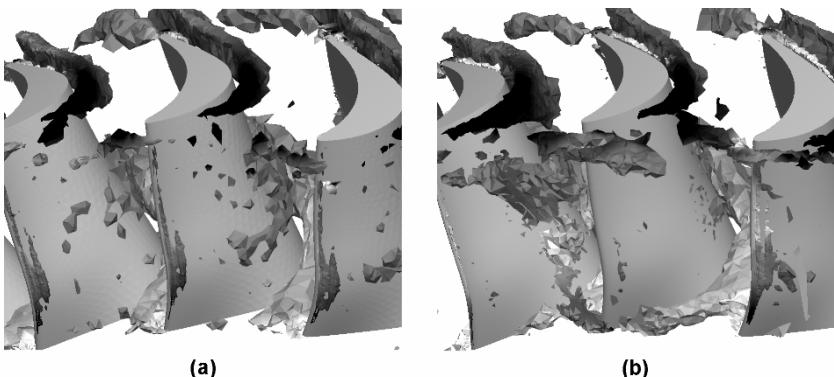


Fig. 5. Vortices in the rotor (vortices coloured by axial vorticity): (a) steady contours from RANS; (b) instantaneous contours from URANS

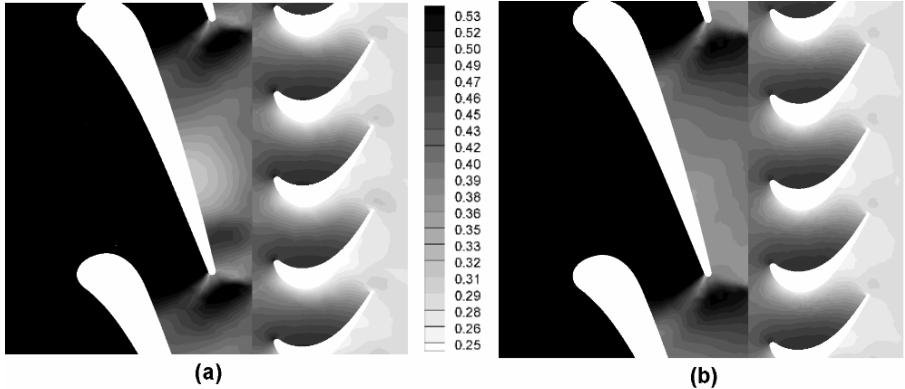


Fig. 6. Iso-contours of static pressure, normalized by the total inlet pressure, at 50% span: (a) steady contours from RANS; (b) time-averaged contours from URANS

Contours of the time-averaged static pressure at 50% span are depicted in Fig. 6. The static pressure in the aft-midsection on the suction side of the vanes from RANS is lower than the corresponding time-averaged pressure from URANS. There is also a sudden increase of static pressure near the trailing edges of the vanes in RANS but not in URANS. All these difference in static pressure are mainly due to the unsteadiness, which is not accounted for in RANS.

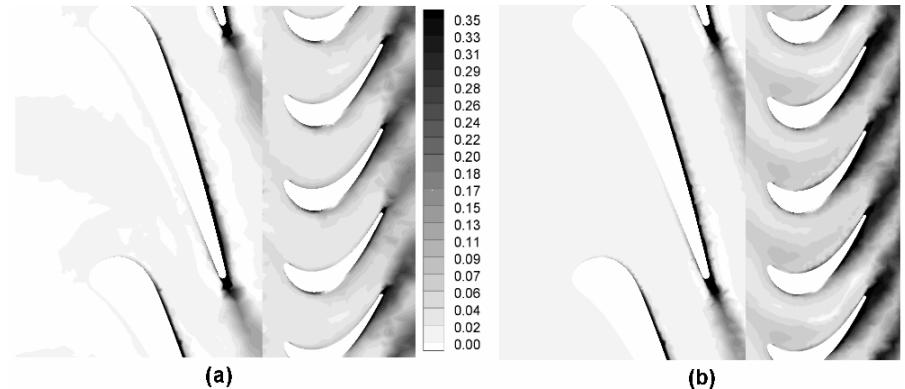


Fig. 7. Iso-contours of entropy, normalized by the gas constant, at 50% span: (a) steady contours from RANS; (b) time-averaged contours from URANS

Figure 7 shows entropy distributions at 50% span for the steady and unsteady simulations. Relatively high entropy indicates high viscous losses and marks boundary layers, wakes and other viscous regions in the flow. While there are almost no differences in the vane wakes within the stator for URANS and RANS, the wakes of the blades are appreciably stronger for the URANS and the losses within the blade passages are also higher in the URANS.

Figure 8 shows isocontours of the total pressure loss coefficient in the rotor reference frame. The relative total pressure loss coefficient at the rotor outlet is defined as

$$Y_r = \frac{\tilde{P}_{02r} - P_{03r}}{\tilde{P}_{03r} - \tilde{P}_3} \quad (2)$$

where the mass-weighted pressure is $\tilde{P} = \int P \rho V dA / \int \rho V dA$ and the relative total pressure is $\tilde{P}_{0r} = \int P_{0r} \rho V dA / \int \rho V dA$.

The total pressure loss from URANS is highest near the midspan of the rotor outlet and decreases towards the endwalls. This high loss may be related to the rotor blade wakes shown in Fig. 8. In contrast, the total pressure losses from RANS show two peaks.

The differences in mass-weighted total pressure coefficients are presented in Table 1, which indicates that the total pressure losses from URANS are measurably higher than the ones from RANS. The percent difference of the relative total pressure loss coefficients at the rotor outlet was 10.9%, which is comparable to the value 10% reported by Pullan [10] for rotor-only simulations.

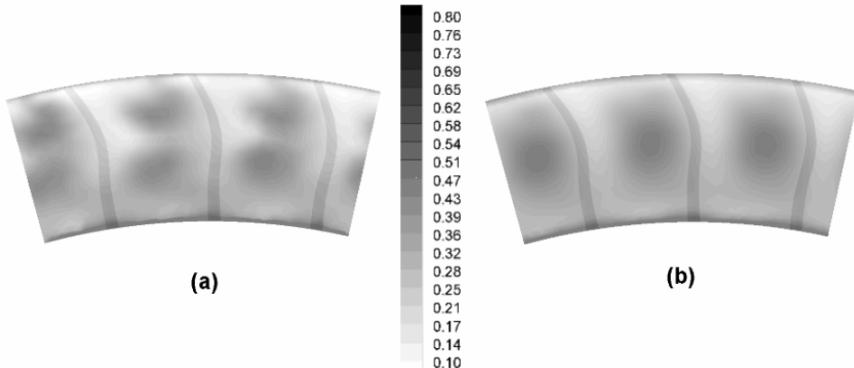


Fig. 8. Iso-contours of relative total pressure loss coefficient on the rotor outlet plane(a) steady contours from RANS; (b) time-averaged contours from URANS; view is towards upstream

Table 1. Comparison of mass-weighted total pressure loss coefficients

	Total pressure loss (Y) at stator outlet	Relative total pressure loss (Y _r) at rotor outlet
RANS	0.6053	0.4477
URANS	0.6257	0.5025
Difference (%)	3.3	10.9

Finally, Fig. 9 shows the static pressure distributions on the blade suction side, determined from RANS and URANS. The URANS results indicate that a local low-pressure region is formed at mid-span toward the leading edge of blades and is isolated from the low-pressure region formed by the two passage vortices; in the RANS results, the three low-pressure regions are connected. It was reported by Chang and Tavoularis [5] that the local low-pressure regions are highly correlated to the locations of blade passage vortices, tip leakage vortices, and crown vortices. Another visible difference is that the pressure in the forward part of the blade tips is stronger for URANS than one for RANS.

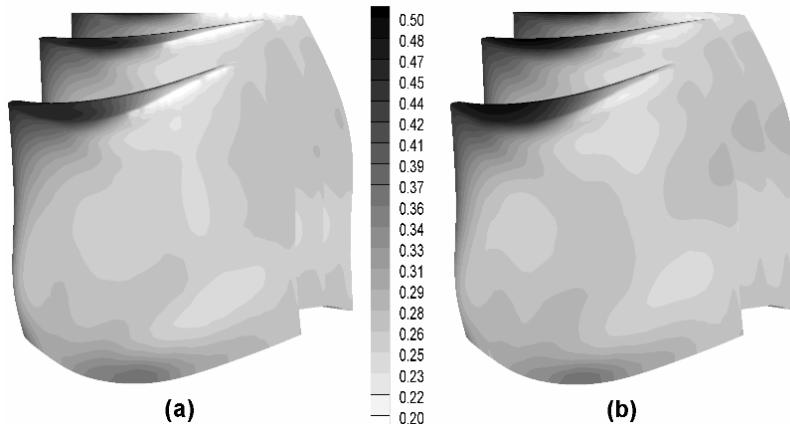


Fig. 9. Iso-contours of static pressure normalized by total inlet pressure: (a) steady contours from RANS; (b) time-averaged contours from URANS

4 Conclusions

Numerical simulations in a high-pressure turbine have been completed. The present article focuses on comparisons of steady-state results and time-averaged results. The main conclusions are as follows.

- The time-averaged static pressure distribution on the suction side of the vanes predicted by URANS was different from the corresponding static pressure distribution in RANS.
- Rotor wakes and total pressure loss at the midspan were stronger in the unsteady simulations.
- The time-averaged static pressure distribution on the blades predicted by URANS was different from the corresponding static pressure distribution in RANS. In the front parts of the blade tips, the static pressure was higher in the unsteady simulations.

By comparison to steady simulations, unsteady simulations have the obvious advantage of resolving the temporal fluctuations of the pressure, velocity and other properties, which can then be used in vibratory stress analyses, an essential aspect of high-pressure turbine design.

Acknowledgment. Financial support for this project has been provided by the Natural Sciences and Engineering Research Council of Canada and by Pratt and Whitney Canada.

References

1. Rai, M.M.: A Direct Numerical Simulation of Transition and Turbulence in a Turbine Stage. In: AIAA 2009-584, 47th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, Orlando, Florida, January 5-8 (2009)
2. Michelassi, V., Wissink, J., Rodi, W.: Analysis of DNS and LES of Flow in a Low Pressure Turbine Cascade with Incoming Wakes and Comparison with Experiments. *Flow, Turbulence and Combustion* 69, 295–330 (2002)
3. Magagnato, F., Pritz, B., Gabi, M.: Calculation of the VKI Turbine Blade with LES and DES. *J. Thermal Sci.* 16, 321–327 (2007)
4. Wu, X., Li, L.T., Hilaire, M.S.: Migration of Turbulent Patch through a High-Pressure Turbine Cascade. *Phys. Fluids* 21, 025110 (2009)
5. Chang, D., Tavoularis, S.: Unsteady Vortices and Blade Loading in a High-Pressure Turbine. In: ASME GT2009-59189, Proceedings of ASME Turbo Expo 2009: Power for Land, Sea and Air, Orlando, Florida, June 8-12 (2009)
6. Menter, F.R.: Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications. *AIAA J.* 32, 1598–1605 (1994)
7. Ganesan, V.: Gas Turbines. Tata McGraw-Hill, New York (2003)
8. Clark, J.P., Grover, E.A.: Assessing Convergence in Predictions of Periodic-Unsteady Flowfields. *ASME J. Turbomach.* 129, 740–749 (2007)
9. Chang, D., Tavoularis, S.: Numerical Simulation of Turbulent Flow in a 37-Rod Bundle. *Nucl. Eng. Des.* 237, 575–590 (2007)
10. Pullan, G.: Secondary Flows and Loss Caused by Blade Row Interaction in a Turbine Stage. *ASME J. Turbomach.* 128, 484–491 (2006)

Application of Parallel Processing to the Simulation of Heart Mechanics

Matthew G. Doyle¹, Stavros Tavoularis¹, and Yves Bourgault^{1,2}

¹ Department of Mechanical Engineering, University of Ottawa, Ottawa, Canada
stavros.tavoularis@uottawa.ca

² Department of Mathematics and Statistics, University of Ottawa, Ottawa, Canada

Abstract. Simulations of the mechanics of the left ventricle of the heart with fluid-structure interaction benefit greatly from the parallel processing power of a high performance computing cluster, such as HPCVL. The objective of this paper is to describe the computational requirements for our simulations. Results of parallelization studies show that, as expected, increasing the number of threads per job reduces the total wall clock time for the simulations. Further, the speed-up factor increases with increasing problem size. Comparative simulations with different computational meshes and time steps show that our numerical solutions are nearly independent of the mesh density in the solid wall (myocardium) and the time step duration. The results of these tests allow our simulations to continue with the confidence that we are optimizing our computational resources while minimizing errors due to choices in spatial or temporal resolution.

Keywords: Biomechanics, heart mechanics, fluid-structure interaction.

1 Introduction

The left ventricle (LV) is the main pumping chamber of a mammalian heart; it is responsible for pumping blood through the body and, by comparison to other parts of the heart, it is more prone to develop malfunction or failure. Simulations of the mechanics of the LV require the calculation of the deformation of the LV wall, the dynamics of the flowing blood, and the interaction between the two, which is referred to as fluid-structure interaction (FSI). Many previous studies of ventricular mechanics have focused on either solid mechanics [1], fluid mechanics [2], or other aspects of the heart simulations, such as electromechanical coupling [3], but have disregarded FSI effects. FSI simulations of the whole heart have been performed by Peskin and McQueen [4] using the immersed-boundary method, but these simulations did not address in detail the wall mechanics. Finite element simulations of the human LV have been performed by Watanabe *et al.* [5], who used simplified assumptions to determine parameter values for their material model, and did not validate their results for the fluid or solid against previous experimental studies.

Although detailed simulations of flow in the LV have been performed using patient-specific geometries generated from magnetic resonance imaging (MRI) techniques [2], the motion of the LV wall was specified using MRI data, rather than

solving appropriate solid mechanics equations. In a recent paper, Ge and Ratcliffe [6] comment on a fluid-only study similar to the one by Long *et al.* [2]; they concede that coupling fluid-only simulations with simulations of the mechanics of the wall would strengthen their results, but do not cite any examples of such coupled finite element simulations that incorporate realistic models for both the fluid and the solid materials. In summary, although a lot of work is being conducted on various aspects of cardiovascular mechanics, the modeling of realistic interactive wall and blood motions remains a relatively unexplored area of research.

One of the challenges of performing FSI simulations in the LV is the requirement of large amounts of RAM and CPU power, which necessitate the use of a high performance computing facility, such as the clusters available to researchers at several universities and colleges in Eastern Ontario, Canada through membership in the High Performance Computing Virtual Laboratory (HPCVL).

In the present study, we consider the LV of a canine heart. We chose the canine heart among those of other animals, because of the availability of geometrical specifications, stress-strain measurements for material model development, and results from previous experimental and computational studies, which can be used for validation and comparisons. Three types of simulations are performed in the present study: development of a material model for the solid wall, simulation of passive LV filling, and simulation of the cardiac cycle. Details on the solid material model have been described elsewhere [7]. Passive LV filling simulations model the inflation of an excised canine LV using a linearly-increasing quasi-static pressure, and are meant to serve as initial conditions for cardiac cycle simulations. Cardiac cycle simulations model the mechanics of the fluid and solid parts of the LV over a single heartbeat. All three sets of simulations in this work are performed using the commercial finite element software ADINA v. 8.5.2 (ADINA R & D, Inc., Watertown, MA, USA) on 64-bit Sun computers running the Solaris operating system.

The objective of this article is to describe the computational requirements for our study in terms of RAM and CPU time, as well as to determine the appropriate number of CPUs for the simulations based on a parallelization study. Other numerical aspects of this work, including some grid and time-step dependence studies, will also be described. To demonstrate the feasibility and limitations of this approach, we will also present some preliminary simulation results.

2 Methods

2.1 Geometries

The heart wall is made up of three layers, which, from the exterior to the interior, are the epicardium, the myocardium, and the endocardium. The epicardium and the endocardium are much thinner than the myocardium and have not been considered in this work. The myocardium is made up of muscle fibres, interstitial fluid, blood vessels, blood, and an extracellular matrix, which contains, among other things, collagen and elastin [8]. The muscle fibres in the myocardium are arranged in layers of roughly constant fibre orientation through the thickness of the wall, where the fibre angle, with respect to the local circumferential direction, in a canine LV varies from +60° in the inner layer to -60° in the outer layer [9].

The geometry used in this study is an idealization of an isolated canine LV, whose overall shape is that of a truncated prolate ellipsoid (i.e., the three-dimensional shape generated by an ellipse that is rotated about its major axis). Therefore, we are making use of an idealized geometry rather than one constructed from medical images or measurements of an individual, excised canine heart, such as those performed by Nielsen *et al.* [10]. This approach avoids the need for detailed measurements of the LV geometry, muscle fibre orientations, material properties, and inflow and outflow conditions, which would be extremely difficult to obtain for a single specific specimen. Moreover, because of the very large variability of the values of many of these parameters from one specimen to another, it may not be advisable to mix values obtained from different specimens. By making use of an idealized LV geometry, and selecting values of the various parameters that correspond to averages, we hope to obtain results that are representative of typical conditions in the canine LV, although not necessarily applicable to any specific individual. Our approach could easily be adapted to the simulation of individual LVs under specific conditions (e.g., abnormal or diseased hearts), provided that appropriate values of all necessary parameters are specified.

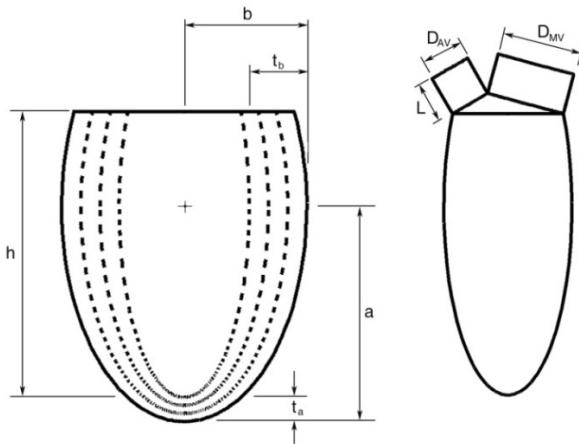


Fig. 1. Solid (left) and fluid (right) parts of the isolated LV geometry

The dimensions of the solid part of the geometry, representing the myocardium, are based on averages of measurements of several dog hearts [11]. The outer semi-major and semi-minor axes, denoted as a and b in Fig. 1, respectively, are 45.2 mm and 25.8 mm, whereas the thicknesses at the apex and the equator are $t_a = 5.1$ mm and $t_b = 12.1$ mm, respectively. The height of the solid geometry from the apex of the inner surface to the flat top surface is $h = 60.2$ mm. The solid wall is evenly subdivided into either three or six layers, each with a specific fibre angle. The fluid geometry consists of two parts. The lower part has an outer boundary which is identical to the inner boundary of the solid geometry to allow for FSI effects to be accounted for along the fluid-solid interface. The upper part of the fluid geometry is rigid and consists of two cylindrical tubes of length $L = 8.7$ mm with diameters $D_{MV} = 16.8$ mm and $D_{AV} = 8.7$ mm, which represent the LV inflow and outflow tracts, respectively; the

inflow tract houses an idealized mitral valve, while the outflow tract houses an idealized aortic valve. The fluid geometry is completed by a section of a sphere, which joins the cylinders to the lower part of the fluid geometry. The fluid and solid parts of the geometry are shown in Fig. 1.

2.2 Meshes

The solid geometry is meshed with an unstructured grid, using ten-node tetrahedral elements, having nodes on each vertex and halfway along each edge. Higher-order elements were chosen for the solid mesh instead of lower-order ones to ensure compatibility with our material model, which requires the use of a mixed interpolation formulation. To prevent singularities at the apex of the LV, the solid geometry had been subdivided into quarters in the vertical direction. Using the three-layer version of the solid geometry to study grid dependence, two mesh densities, which will be referred to as “coarse” and “fine”, have been considered, consisting of 16 861 elements and 25 159 nodes, and 103 411 elements and 148 328 nodes, respectively.

The fluid geometry is meshed on an unstructured grid, using four-node tetrahedral elements. These elements have nodes on each vertex for velocity and pressure, and an additional node at the centre to calculate velocity and ensure stability of the solution. These elements are generally referred to as MINI elements and the velocity at the centre is called the bubble velocity [12].

To study grid dependence, three mesh densities have been considered for the fluid geometry, consisting of 28 103 elements and 5480 nodes, 193 372 elements and 34 652 nodes, and 351 378 elements and 71 915 nodes, respectively. These mesh densities will be referred to as “coarse”, “medium”, and “fine”, respectively. The FSI coupling used in ADINA does not require coincident meshes at the FSI interface, allowing the meshes for both the solid and fluid to be refined or coarsened separately. Figure 2 shows the coarse and fine meshes for the solid and fluid models.

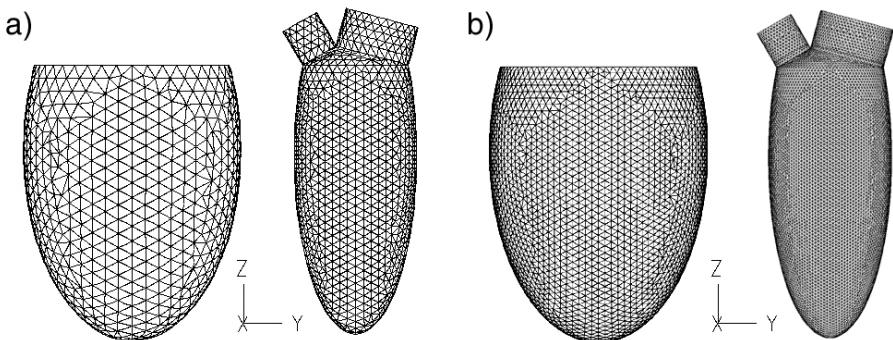


Fig. 2. Solid (*left*) and fluid (*right*) meshes with the coarse (a) and fine (b) mesh densities

2.3 Governing Equations

The LV undergoes large deformations, so the appropriate governing equations are those for non-linear analysis. Both static (solid-only) and dynamic (FSI) simulations

are performed in this work, so both sets of equations are utilized. In the static case, the governing equation is

$$\mathbf{R}(t + \Delta t) - \mathbf{F}(t + \Delta t) = 0 , \quad (1)$$

where \mathbf{R} is the external load vector, and \mathbf{F} is the “force vector equivalent to the element stresses” [13].

In the dynamic case, the governing equation is

$$\mathbf{M}\ddot{\mathbf{U}}(t + \Delta t) + \mathbf{C}\dot{\mathbf{U}}(t + \Delta t) + \mathbf{K}(t)[\mathbf{U}(t + \Delta t) - \mathbf{U}(t)] = \mathbf{R}(t + \Delta t) - \mathbf{F}(t) , \quad (2)$$

where \mathbf{M} is the mass matrix, \mathbf{C} is the damping matrix, \mathbf{K} is the stiffness matrix, and $\ddot{\mathbf{U}}$, $\dot{\mathbf{U}}$, and \mathbf{U} are the nodal accelerations, velocities, and displacements, respectively [13].

The stress-strain behaviour of the myocardium is highly non-linear and was simulated with the use of a user-defined material model, written in the form of a FORTRAN subroutine that was linked with ADINA. This model is based on one proposed by Lin and Yin [14] and has been described in detail elsewhere [7]. It is defined by a strain energy density function, W whose derivatives with respect to the invariants of Green's strain tensor are used to calculate stresses from strains. Although fully-orthotropic material models for the myocardium have been considered in the literature [1], there are no published stress-strain results that would allow for the validation of such models. Instead, our model of the myocardium is a transversely isotropic material with properties that differ in the fibre and cross-fibre directions; adequate experimental data are available for the calculation of the parameters that define this model. Lastly, we have assumed the myocardium to be a slightly compressible material to allow for model convergence of a solid geometry that undergoes large deformations.

During the cardiac cycle, the stress-strain behaviour of the myocardium changes as the muscle fibres contract and relax. More specifically, the stresses in the wall increase as the muscle fibres contract and decrease as the muscle fibres relax. To account for this change in stress-strain behaviour, the material model is defined in two parts, a passive part W_p , which represents the stress-strain behaviour of the myocardium when the muscle fibres are fully relaxed, and an active part W_a , which represents the additional stresses from the fully contracted muscle fibres that are added to the passive part to give the total stress state. These two parts of the material model are defined as

$$W_p = C_1(e^Q - 1) + \frac{1}{2}\kappa_s(J_3 - 1)^2 , \quad (3)$$

$$Q = C_2(J_1 - 3)^2 + C_3(J_1 - 3)(J_4 - 1) + C_4(J_4 - 1)^2 + C_5(J_1 - 3) + C_6(J_4 - 1) , \quad (4)$$

$$W_a = D_0 + D_1(J_1 - 3)(J_4 - 1) + D_2(J_1 - 3)^2 + D_3(J_4 - 1)^2 + D_4(J_1 - 3) + D_5(J_4 - 1) , \quad (5)$$

where C_i and D_i are appropriate material parameters [7], κ_s is the material bulk modulus, and J_i are reduced invariants of Green's strain tensor. Note that the chosen value of κ_s was sufficiently large for the compressibility effects to be negligible.

To model the dynamic contraction and relaxation of the muscle fibres, the active material parameter values D_i are defined as

$$D_i = FD_{i,\max} , \quad (6)$$

where F is a forcing function that varies from 0, when the muscle fibres are fully relaxed to 1, when the muscle fibres are fully contracted, and $D_{i,\max}$ are the values of the active material parameters when the muscle fibres are fully contracted [5]. An example of such a forcing function is shown in Fig. 3. By adjusting the slopes and durations of the ascending and descending parts of the forcing function, we could adjust the rates of muscle fibre contraction and relaxation such as to predict changes in the LV cavity pressure during the cardiac cycle that were as closely as possible to physiological values.

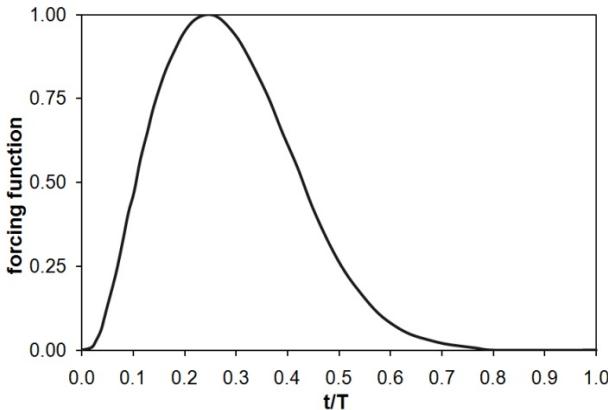


Fig. 3. Forcing function for active material parameter values proposed by Watanabe *et al.* [5]

In the present study, blood is assumed to be a slightly compressible Newtonian fluid, and the flow in the LV is assumed to be laminar. The choice of a slightly compressible fluid was made to allow for model convergence during the parts of the cardiac cycle when both valves are closed, namely during isovolumetric contraction and isovolumetric relaxation. During these times, the fluid domain is entirely enclosed, although it is also deformable. Many finite element codes, including ADINA, cannot enforce strict conservation of volume in an enclosed deformable geometry. To prevent the fluid or solid part of the model from compressing the other during FSI simulations, the bulk moduli of the fluid and solid materials are set to be equal. To account for the deformable fluid mesh, the appropriate governing equations are the slightly compressible, arbitrary-Lagrangian-Eulerian (ALE) form of the continuity and momentum equations, defined as

$$\frac{\rho_f}{\kappa_f} \left(\frac{\partial p}{\partial t} + (\mathbf{v} - \mathbf{w}) \cdot \nabla p \right) + \rho_m \nabla \cdot \mathbf{v} = 0 , \quad (7)$$

$$\rho_f \frac{\partial \mathbf{v}}{\partial t} + \rho_f (\mathbf{v} - \mathbf{w}) \cdot \nabla \mathbf{v} - \nabla \cdot \boldsymbol{\tau} = \mathbf{f}^B , \quad (8)$$

where ρ_f is the fluid density at zero pressure, κ_f is the fluid bulk modulus, p is pressure, t is time, \mathbf{v} is the velocity vector, \mathbf{w} is the mesh velocity vector, $\boldsymbol{\tau}$ is the stress tensor, \mathbf{f}^B is body force per unit volume, and ρ_m is the compressible density [15], defined as

$$\rho_m = \rho_f \left(1 + \frac{p}{\kappa_f} \right) , \quad (9)$$

Based on the assumption of a Newtonian fluid, the stress tensor $\boldsymbol{\tau}$ is defined as

$$\boldsymbol{\tau} = -p \mathbf{I} + \mu (\nabla \mathbf{v} + \nabla \mathbf{v}^T) , \quad (10)$$

where μ is the fluid viscosity and \mathbf{I} is the identity matrix.

2.4 Boundary Conditions

FSI boundary conditions are applied to the inner surfaces of the solid geometry and the outer surfaces of the lower part of the fluid geometry. In ADINA, the following two boundary conditions, one kinematic and one dynamic, are enforced at the FSI boundaries

$$\underline{\mathbf{d}}_f = \underline{\mathbf{d}}_s , \quad (11)$$

$$\underline{\mathbf{n}} \cdot \underline{\boldsymbol{\tau}}_f = \underline{\mathbf{n}} \cdot \underline{\boldsymbol{\tau}}_s , \quad (12)$$

where $\underline{\mathbf{d}}$ is the displacement vector, the subscripts f and s represent the fluid and solid parts of the model, respectively, and the underlines represent the FSI boundary [15]. The fluid velocity at the FSI boundary is calculated from Eq. 11 and the solid model and the force on the solid at the FSI boundary is calculated from Eq. 12 and the fluid model using the following equations

$$\underline{\mathbf{v}}_f = \dot{\underline{\mathbf{d}}}_s , \quad (13)$$

$$\underline{\mathbf{F}}_s(t) = \int \mathbf{H}^{S^T} \underline{\boldsymbol{\tau}}_f \cdot dS , \quad (14)$$

where \mathbf{H}^S contains the element shape functions [16].

The solid geometry is anchored by fixing the top plane in the vertical direction and its inner edge in all three directions. For the rigid part of the fluid geometry, no-slip wall boundary conditions are applied to all outer surfaces. To control the flow direction, the valves in this study, located near the proximal ends of the inflow and outflow

tracts, are modelled as instantly opening and closing surfaces, which are implemented in ADINA using “gap” boundary conditions. Time functions are used to specify that each gap opens or closes at appropriate times during the cardiac cycle, with closed gap boundary conditions acting as no-slip walls.

For passive LV filling simulations, the LV is pressurized from 0 to 2 kPa by a linearly-increasing pressure function, in which the final pressure of 2 kPa corresponds to the end-diastolic pressure proposed by Sabbah and Stein [17]. This end-diastolic pressure value was chosen so that the final state of the passive LV filling simulations can be used as the initial condition for the cardiac cycle simulations. For solid-only simulations, the pressure function is applied to the inner walls of the LV cavity, whereas, for the FSI simulations, it is applied at the distal end of the LV inflow tract. Solid-only simulations are performed statically, but FSI simulations are performed dynamically. To approximate static conditions, the time step for the FSI simulations is chosen to be so long that the fluid velocities become negligible and the cavity volume changes for the dynamic FSI simulations approach the cavity volume changes for the static solid-only simulations.

The cardiac cycle is divided into four phases: isovolumetric contraction, during which both valves are closed and the muscle fibres are contracting; ejection, during which the aortic valve is open, the muscle fibres are contracting, and blood is exiting the LV; isovolumetric relaxation, during which both valves are closed and the muscle fibres are relaxing; and filling, during which the mitral valve is open, the muscle fibres are relaxing, and blood is entering the LV. In the current simulations, the relative durations of the four phases are $t / T = 0.055, 0.257, 0.073$, and 0.615 . During the ejection and filling phases, physiological time-varying left atrial and aortic pressure functions [17] are applied to the distal ends of the inflow and outflow tracts, respectively, corresponding to times when each valve is open.

2.5 Numerical Methods

For FSI simulations in ADINA, time stepping is controlled by the fluid model. For the current simulations, a second-order time stepping method, referred to as the ADINA composite method, is used. This method consists of two sub-time steps, which allows for fewer time steps to be used than those required by the available first-order method, which does not have sub-time steps. This approach leads to a reduction in overall computational time. The ADINA composite time integration method is defined as

$$\begin{aligned} u(t + \gamma\Delta t) &= u(t) + \gamma\Delta t f(u(t + \frac{1}{2}\gamma\Delta t)) \\ u(t + \Delta t) &= u(t + \beta\gamma\Delta t) + (1 - \alpha)\Delta t f(u(t + \Delta t)) \end{aligned}, \quad (15)$$

where $u(t + \beta\gamma\Delta t) = (1 - \beta)u(t) + \beta u(t + \gamma\Delta t)$, $\gamma = 2 - \frac{1}{\alpha}$, $\beta = \frac{\alpha^2}{(2\alpha - 1)}$, and $\alpha = \frac{1}{\sqrt{2}}$ [15].

In this study, FSI simulations are conducted using a direct solver, in which the fluid and solid equations, along with those on the FSI boundaries, are combined into a single matrix and solved simultaneously. An alternative approach would have been the use of an iterative solver, in which the fluid and solid parts of the model are solved sequentially, with information passed between them on the FSI boundaries. Nevertheless, the direct solver approach was chosen because the iterative solver proved to be unstable, leading to a divergent solution during the first time step. Beside

its advantage in reaching convergence, the direct solver yields, in general, faster results than the iterative solver [15]. On the other hand, the direct solver requires more RAM than the iterative solver, because, unlike the latter, the former requires all equations to reside in RAM at any given time.

For the present FSI simulations, using the direct solver with the coarse solid mesh requires approximately 3.1 GB of RAM for the coarse fluid mesh, 6.7 GB of RAM for the medium fluid mesh, or 12.4 GB of RAM for the fine fluid mesh, with the majority of this RAM needed for the fluid model. Simulations in this study were performed on HPCVL's Sun SPARC Enterprise M9000 Servers, each consisting of 64 quad-core Sparc64 VII 2.52 GHz processors, which are capable of running 2 threads per processor.

3 Results and Discussion

3.1 Parallelization

To study the effects of increasing the number of threads on wall clock time, an analysis of the parallelization performance was carried out for FSI simulations of passive LV filling using the three-layer geometry, the coarse solid mesh, and the three different fluid meshes. Simulations were performed using 2, 4, 8, 16, and 32 threads. The resulting wall clock times were compared to the wall clock time for a single thread to calculate the speed-up factor (SU), defined as the wall clock time for the single-thread simulations divided by the wall clock time for the multi-thread simulations.

The results of the calculations of the speed-up factor are presented in Fig. 4, along with the ideal speed-up, which is equal to the number of threads. As expected, the actual speed-up factor increases with increasing number of threads, and with increasing mesh density. Moreover, the differences between the actual and ideal results increase with increasing number of threads.

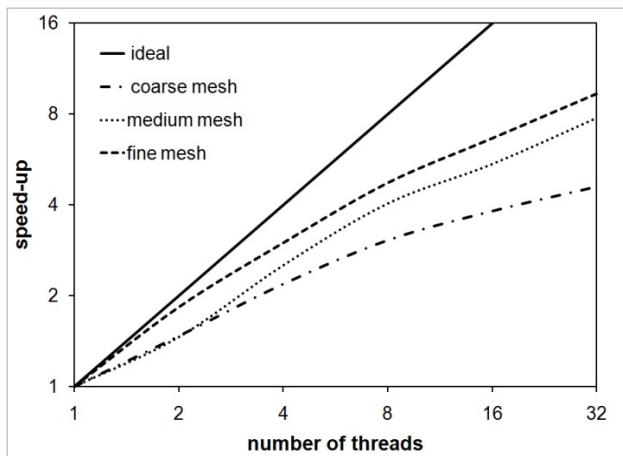


Fig. 4. Speed-up of parallel FSI simulations with three difference fluid mesh densities versus the number of threads, plotted in logarithmic axes

To understand why the speed-up factors are much less than ideal, we will examine the parallelization procedure in ADINA. This information will be used to determine the theoretical maximum speed-up factor given by Amdahl's law

$$SU_{\max} = \frac{1}{(1-P) + (P/N)} , \quad (16)$$

where P is the fraction of the code that is parallelized and N is the number of threads.

To calculate P , the total computational time is divided into the parts that are serial and the parts that are parallel. An example of the breakdown of the computational time for one time step with the fine fluid mesh and 1 or 32 threads is given in Table 1. In ADINA, only the sparse solver, which is a direct solver used to solve both the fluid and solid equations, is parallelized [13]. As will be shown in the following, the solid solver can be assumed to be serial, so that only the fluid solver needs to be considered as parallel. Although there are differences in the CPU times from 1 to 32 threads for the "serial components" program control, solid model, fluid assembly, moving fluid mesh, and fluid output, the sum of these components is nearly constant (848.00 s for 1 thread and 838.91 s for 32 threads), even though the solid solver is also parallelized. In view of the relatively small change in CPU time for these components, we will assume them to be serial for the purposes of calculating SU_{\max} . P can then be calculated from the results for 1 thread as the CPU time for the fluid solver divided by the total CPU time. Using this value of P , SU_{\max} can be calculated for 2, 4, 8, 16, and 32 threads; an upper limit for SU_{\max} can be also calculated by letting $N \rightarrow \infty$. Similarly, P can be found for the other two mesh densities. The results of the calculations of SU_{\max} are presented in Table 2.

Table 1. CPU time for the parts of the simulations with the fine fluid mesh

Simulation part	CPU time (s), $N = 1$	CPU time (s), $N = 32$
program control	66.50	44.80
solid model	153.25	173.25
fluid assembly	497.12	560.12
fluid solver	25274.12	1947.25
moving fluid mesh	115.88	44.62
fluid output	15.25	16.12

Table 2. Maximum speed-up factors for the three fluid mesh densities

Number of threads	Coarse	Medium	Fine
2	1.8	1.9	1.9
4	2.9	3.5	3.6
8	4.2	6.1	6.5
16	5.5	9.5	10.8
32	6.5	13.3	15.9
∞	7.9	21.9	30.8

We now provide some insight on how the parallel fraction P varies for these FSI simulations. Table 1 shows that, for our FSI simulations, the solution of the solid part of the model requires a small fraction of the total computational time, while the fluid solver uses by far the largest portion of this time. This comes from the fact that meshes for simulating flows need to be much finer than meshes used in structural mechanics, which can be coarse and fixed, as we will show in the following. Moreover, for FSI simulations, the coupling of the motions of the fluid and solid boundaries forces repeated flow computations which could, in the worst case, amount to solving the same flow multiple times on a fixed geometry. These facts imply that, in an effort to parallelize a FSI code, one must first act on the fluid solver. As the fluid mesh is refined while the solid mesh remains fixed, the parallel fraction P of the code would increase. This is demonstrated by our calculations of P , which were 0.87, 0.95 and 0.97 for our test cases with the coarse, medium, and fine fluid meshes, respectively.

A comparison of the results in Table 2 with those in Fig. 4 shows that the maximum speed-up factors deduced from Amdahl's law are smaller than actual speed-up factors obtained from computations. Indeed these maximum speed-up factors do not account for the increase in communication time with growing number of threads and cannot be achieved in practice. For example, with 32 threads on the fine mesh, SU_{\max} is about 16, whereas the speed-up factor observed in practice is only about 9. As for any parallel computations, additional threads improve the performance of our FSI simulations, and even more so with growing problem size, but the large amounts of memory and inter-process communications required for FSI severely limit the scalability of these parallel simulations. We used the direct fluid-structure solution method from ADINA. This method reduces the number of flow computations compared to the iterative fluid and solid solver described in Section 2.6, but at the expense of requiring a larger memory and more extensive communications, which combined with a reduced parallelized fraction P explains our relatively poor speed-up obtained with a large number of threads.

In spite of their poor performance, parallel computations are necessary for FSI as they reduce the total computational time, even if by much less than an ideal case. For example, in our passive LV filling simulations with the coarse fluid mesh, the total computational time was reduced from 120.4 h for 2 threads to 34.0 h for 32 threads. This impact is more significant when considering that for the medium and fine fluid mesh densities, passive LV filling simulations take 119.3 h and 254.4 h, respectively, with 32 threads. The computational time for the full cardiac cycle simulations is considerably longer than for the passive LV filling, which makes parallelization even more essential.

The computational times for these simulations should be viewed as lower bounds on the computational times needed for more complete simulations. Additional fluid mesh refinement, coupling of the LV geometry with other parts of the cardiovascular system, and/or the use of a patient-based LV geometry would all add substantially to the computational time required for simulations. This is clearly one of the reasons why many researchers have neglected FSI effects in their heart simulations and why performing FSI simulations in the heart remains a challenging computational problem.

3.2 Grid Dependence

For the solid model, solid-only passive LV filling simulations are used to study grid dependence, whereas for the fluid model, FSI simulations of the cardiac cycle will be considered. The fluid grid dependence simulations were performed over the cardiac cycle instead of during the passive LV filling because the passive LV filling simulations are performed such that the fluid velocity approaches zero, which implies that grid dependence issues may not be evident in the fluid mesh.

Both global and local indicators are used to examine grid dependence in the solid model. Figure 5 shows the change in cavity volume as a function of pressure, which represents a global indicator. Figure 6 shows a local indicator of the results, namely, the principal stretches as a function of change in cavity volume. These principal stretches were taken in the middle layer of the wall at a distance halfway between the apex and the base of the LV, and are the averages of values at three circumferential locations corresponding to the range of locations that make up what would be the LV free wall if the LV were attached to a right ventricle. These locations were chosen to roughly match the locations of measurements in a previous study [18].

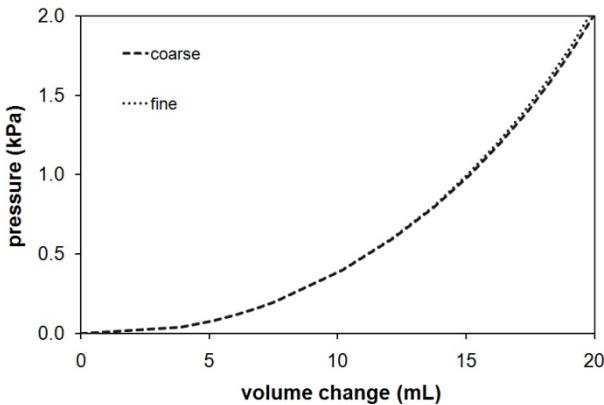


Fig. 5. LV pressure versus change in cavity volume for two different solid mesh densities

The pressure-volume results shown in Fig. 5 for the coarse and fine solid meshes are nearly coincident, as small differences become visible only at the largest volumes. At 2 kPa, the difference between the results from the two meshes is less than 1 %. For the principal stretches in Fig. 6, a small difference can be seen between the results from the two meshes, particularly in λ_1 and λ_2 . At a normalized volume change of approximately 20 mL, the difference between the principal stretches for the coarse and fine meshes is less than 1 % in all three directions. Because the differences in the results from the two mesh densities are so small, both locally and globally, it is concluded that the coarse mesh provides adequate solution accuracy; consequently, the coarse solid mesh will be used for the remainder of the computations.

The temporal variations of the pressure in the fluid at the centre of the top plane of the ellipsoidal section of the fluid geometry and the LV cavity volume for the cardiac cycle simulations are presented in Figs. 7 and 8, respectively. These simulations are

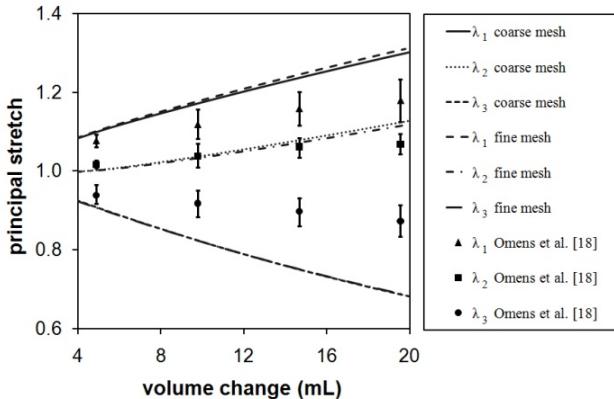


Fig. 6. Principal stretches versus volume change in the middle layer of the myocardium half-way between the base and the apex. Current results are averages of three circumferential locations. Points are from a previous experimental study [18].

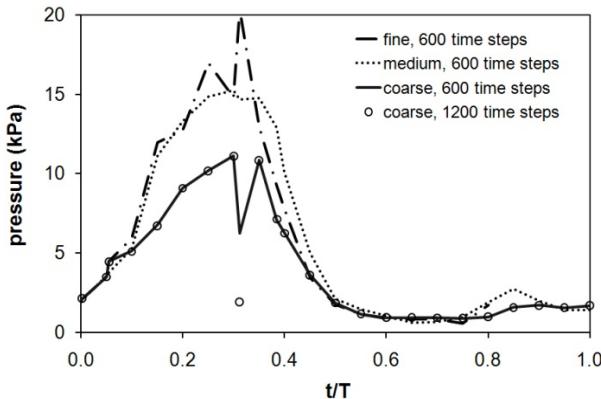


Fig. 7. Pressure variation at the centre of the top plane of the ellipsoidal fluid geometry during the cardiac cycle

still in progress in an effort to achieve physiologically realistic results; they require optimization of model parameters and adjustment of the forcing function used to control the transition between the passive and total stress states. Even so, the state of simulations is sufficiently advanced for the evaluation of the effects of mesh density and time step to be possible.

Before discussing the grid dependence of the results in Figs. 7 and 8, it would be helpful to clarify some issues concerning the ongoing simulations. The forcing function used in the simulations whose results are presented in these figures does not produce sufficiently large pressure during isovolumetric contraction to allow blood to exit the LV during ejection. In fact, as shown in Fig. 8, blood is driven into the LV cavity, causing the cavity volume to increase during a time that should correspond to the ejection phase. Similarly, the LV cavity volume decreases during the first part of the filling phase. This problem is expected to be corrected in ongoing simulations. Its

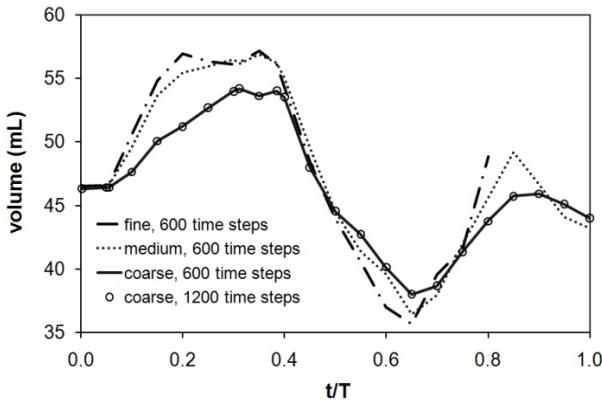


Fig. 8. LV cavity volume variation during the cardiac cycle

presence, however, is believed to be immaterial to the determination of grid independence. In addition to the issues with backflow, it can be noted from Fig. 8 that the volume at $t / T = 1$ differs from the volume at $t / T = 0$. The cardiac cycle simulations are periodic, and will most likely require several periods to approach a periodic state that is independent of the initial conditions. If such a state is achieved in the simulations, the LV cavity volume would be conserved from one period to the next. Lastly, it should be noted that simulations with the fine mesh density were only performed until $t / T = 0.8$ because of an unexpected computer system failure.

Comparison of the results in Figs. 7 and 8 for the three fluid mesh densities clearly show that the coarse fluid mesh is incapable of predicting accurately pressure and volume changes. The predicted pressure and volume changes for the medium and fine mesh densities show fairly good agreement, with the exception of the two spikes in pressure with the fine mesh density near $t / T = 0.3$. The second spike corresponds to the opening of the aortic valve, which led to a large drop in pressure for simulations with the coarse mesh density. It is hypothesized that with the increase in mesh density from the medium to the fine mesh, the number of time steps required to properly resolve the pressure has also increased. Additional simulations are required to test this hypothesis. For now, given the large increase in computational time from the medium to the fine fluid meshes, the medium mesh is deemed to give results that are sufficiently accurate to be used in our ongoing cardiac cycle simulations that serve to adjust the model parameters to obtain a physiological solution. It should be noted that although the solution has clearly not yet reached a periodic state, by starting from the same initial conditions in all cases, the results in Figs. 7 and 8 show that the transient solution is converging as the fluid mesh is refined, which in turn implies that, if grid independence is reached in this case, it should also apply to the periodic solution.

3.3 Time Step Dependence

Results for the two time step durations shown in Figs. 7 and 8 are in excellent agreement, with the exception of a difference in pressure at $t / T = 0.31$ in Fig. 7, which

signifies the opening of the aortic valve and the start of the ejection phase. The aortic valve opens over a single time step, so, because the time step durations are different in the two cases, the lengths of time over which the valve opens are different. The differences in pressure at this single time are caused by this difference in the duration of the valve closure, and are not suggestive of the need for further decreases in the time step size, especially if one considers the agreement between the two cases over the rest of the cycle. It is concluded that using 600 time steps is sufficient for simulations with this mesh density. However, additional time steps may be required for finer meshes.

3.4 Passive Left Ventricle Filling Results

Principal stretches for the current simulations, presented in Fig. 6, have comparable trends but slightly larger magnitudes than the previous measurements of Omens *et al.* [18]. Differences in magnitudes could be due to differences in initial LV cavity volumes, which were 26.7 mL for the current geometry and 19.4 mL for the previous one, or masses of the LV wall, which were 80.42 g for the current geometry, assuming a density of 1060 kg/m³ [19], and 97.80 g for the previous geometry. However, the most significant factor in the differences in magnitudes between the current and previous results may be the enforcement of the assumption that the myocardium is a slightly compressible material. Enforcement of this condition implies that the third invariant of Green's strain tensor I_3 , which is the product of the squares of the three principal strains ($I_3 = \lambda_1^2 \lambda_2^2 \lambda_3^2$), should be close to one for all volume changes. For the current results, at a volume change of 19.96 mL, $I_3 = 1.003$, while for the previous results, at a volume change of 19.56 mL, $I_3 = 1.21$. While the components of the myocardium, muscle fibres, interstitial fluid, blood vessels, blood, and extracellular matrix, can all be assumed to be nearly incompressible, the transfer of fluid into and out of the blood vessels of the myocardium during passive LV filling leads to changes in overall volume, which implies that the myocardium behaves as a compressible material [20]. To properly account for this compressibility, the inclusion of poroelastic and viscoelastic models, such as those proposed by Huyghe *et al.* [8] should be considered. However, these models are beyond the scope of the present study.

3.5 Preliminary Cardiac Cycle Results

Cardiac cycle results for the current model are not yet ready for presentation as it has been recognized that the forcing function for the active material model needs further adjustment to prevent the occurrence of unrealistic flow patterns and stress distributions. To serve as examples of the type of results that we expect to present when this study is complete, Figs. 9 and 10 show blood velocity vector maps and bands of effective stress for the myocardium wall for preliminary cardiac cycle simulations at $t / T = 1/6$ and $3/4$, which correspond to mid-ejection and mid-filling, respectively. These results were obtained using an earlier version of the geometry, which differs from the current one in diameters and angles of the inflow and outflow tracts, the placements of the valves, the shape of the spherical cap, the size and thickness of the LV wall, and several model parameters. Changes in the geometry are certain to produce changes in the flow patterns and in the stress distribution in the myocardium. Moreover, the preliminary simulations also suffer from additional inconsistencies, which

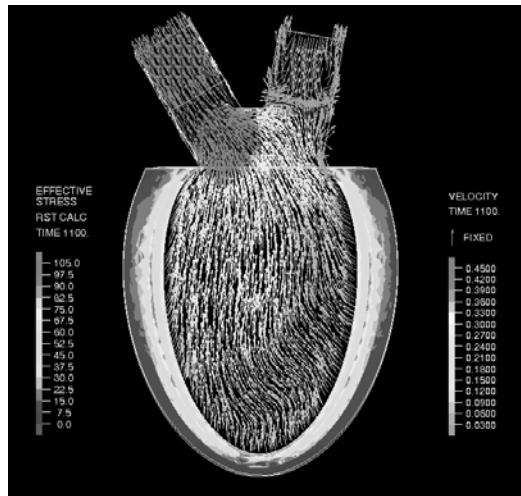


Fig. 9. Blood velocity vectors and myocardium effective stresses during mid-ejection ($t = T/6$)

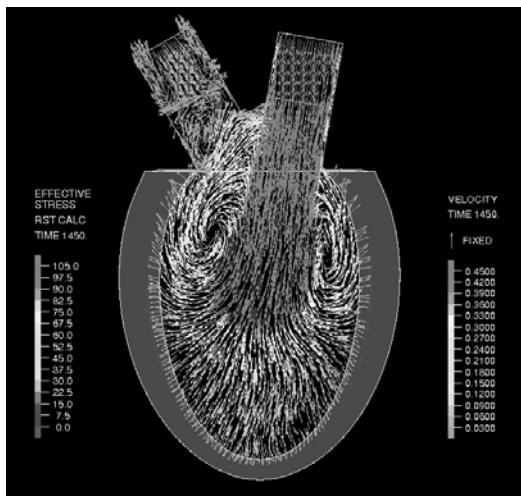


Fig. 10. Blood velocity vectors and myocardium effective stresses during mid-filling ($t = 3T/4$)

we are in the process of correcting. For example, some backflow has been observed during parts of the cycle (not shown in the figures) at which physiological conditions would produce forward flow. Although the flow patterns in the two images shown here are compatible qualitatively with intuitive expectations, it also turned out that the rate of change of LV volume during much of the cycle was significantly lower than the physiological rate. To match the physiological volume change rate, we are currently devising and testing different forcing functions for the active material.

4 Conclusions

We have demonstrated by example and by analysis that parallelization is essential to finite element simulations of the mechanics of the LV with fluid-structure interaction. In particular, we have shown the importance of parallelization of the fluid solver as this account for the majority of computational time in our FSI simulations; more specifically, for simulations on a single thread, the fluid solver requires more than 87 % of the computational time. Previous simulations of LV mechanics neglecting FSI effects avoid the significant increase in computational time required by the fluid part of the model when multiple iterations between the fluid and the solid are performed, but by doing so they miss an important aspect of the problem. We have shown that the speed-up factor increases with increasing number of threads and with increasing mesh density. Even though our speed-up factors are significantly lower than the theoretical maxima for a given problem size and number of threads, we were still able to achieve significant reductions in computational time, which would become increasingly important as we move from passive LV filling to cardiac cycle simulations, or if we were to introduce further geometric complexity to the model.

Grid dependence studies for the solid model showed little difference between the results for two different mesh densities, suggesting that our solution is nearly independent of solid mesh density. Grid dependence studies for the fluid model showed minor differences between the medium and fine mesh densities, which may be attributed to a need to increase the number of time steps for simulations with the fine mesh. However, following consideration of the much longer computational time required for the fine mesh, the medium mesh was deemed to be adequate for our current cardiac cycle simulations.

Results of passive LV filling simulations are in fair agreement with previous measurements, which gives us confidence to use these simulations as initial conditions for cardiac cycle simulations.

The use of high performance computing clusters, such as those available through HPCVL, will enable continuing improvements in the simulation of heart mechanics, by allowing additional details, such as an anatomically realistic geometry, to be incorporated into heart models, while still allowing researchers to obtain simulation results within acceptable time limits.

Acknowledgments

Funding for this work has been provided by two Ontario Graduate Scholarships in Science and Technology (OGSST) to the first author and by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Nash, M.P., Hunter, P.J.: Computational Mechanics of the Heart. *J. Elasticity* 61, 113–41 (2000)
2. Long, Q., Merrifield, R., Xu, X.Y., Kilner, P., Firmin, D.N., Yang, G.-Z.: Subject- Specific Computational Simulation of Left Ventricular Flow Based on Magnetic Resonance Imaging. *Proc. IMechE Part H J. Eng. Med.* 222, 475–485 (2008)

3. Chapelle, D., Fernández, M.A., Gerbeau, J.-F., Moireau, P., Sainte-Marie, J., Zemzemi, N.: Numerical Simulation of the Electromechanical Activity of the Heart. In: Ayache, N., Delingette, H., Sermesant, M. (eds.) FIMH 2009. LNCS, vol. 5528, pp. 357–365. Springer, Heidelberg (2009)
4. Peskin, C.S., McQueen, D.M.: Fluid Dynamics of the Heart and its Valves. In: Othmer, H.G., Adler, F.R., Lewis, M.A., Dallon, J.C. (eds.) Case Studies in Mathematical Modeling – Ecology, Physiology, and Cell Biology, pp. 309–337. Prentice-Hall, Englewood Cliffs (1996)
5. Watanabe, H., Sugiura, S., Kafuku, H., Hisada, T.: Multiphysics Simulation of Left Ventricular Filling Dynamics using Fluid-Structure Interaction Finite Element Method. *Bioophys. J.* 87, 2074–2085 (2004)
6. Ge, L., Ratcliffe, M.: The Use of Computational Flow Modeling (CFD) to Determine the Effect of Left Ventricular Shape on Blood Flow in the Left Ventricle. *Ann. Thorac. Surg.* 87, 993–994 (2009)
7. Doyle, M.G., Tavoularis, S., Bourgault, Y.: Adaptation of a Rabbit Myocardium Material Model for Use in a Canine Left Ventricle Simulation Study. *J. Biomech. Eng.* (in press, 2010)
8. Huyghe, J.M., Van Campen, D.H., Art, T., Heethaar, R.M.: A Two-Phase Finite Element Model of the Diastolic Left Ventricle. *J. Biomech.* 24, 527–538 (1991)
9. Streeter Jr., D.D., Spotnitz, H.M., Patel, D.P., Ross Jr., J., Sonnenblick, E.H.: Fiber Orientation in the Canine Left Ventricle During Diastole and Systole. *Circ. Res.* 24, 339–347 (1969)
10. Nielsen, P.M.F., LeGrice, I.J., Smaill, B.H., Hunter, P.J.: Mathematical Model of Geometry and Fibrous Structure of the Heart. *Am. J. Physiol.* 260, H1365–H1378 (1991)
11. Streeter Jr., D.D., Hanna, W.T.: Engineering Mechanics for Successive States in Canine Left Ventricular Myocardium: I. Cavity and Wall Geometry. *Circ. Res.* 33, 639–655 (1973)
12. Gresho, P.M., Sani, R.L.: Incompressible Flow and the Finite Element Method. Isothermal Laminar Flow, vol. 2. John Wiley and Son, Ltd., Chichester (1998)
13. ADINA R & D, Inc.: ADINA Theory and Modeling Guide Volume I: ADINA. ADINA R & D, Inc., Watertown, MA, USA (2008)
14. Lin, D.H.S., Yin, F.C.P.: A Multiaxial Constitutive Law for Mammalian Left Ventricular Myocardium in Steady-State Barium Contracture or Tetanus. *J. Biomech. Eng.* 120, 504–517 (1998)
15. ADINA R & D, Inc.: ADINA Theory and Modeling Guide Volume III: ADINA CFD & FSI. ADINA R & D, Inc., Watertown, MA, USA (2008)
16. Zhang, H., Bathe, K.-J.: Direct and Iterative Computing of Fluid Flows Fully Coupled With Structures. In: Bathe, K.-J. (ed.) Computational Fluid and Solid Mechanics, pp. 1440–1443. Elsevier Science Ltd., Oxford (2001)
17. Sabbah, H.N., Stein, P.D.: Pressure-Diameter Relations During Early Diastole in Dogs: Incompatibility with the Concept of Passive Left Ventricular Filling. *Circ. Res.* 45, 357–365 (1981)
18. Omens, J.H., May, K.D., McCulloch, A.D.: Transmural Distribution of Three-Dimensional Strain in the Isolated Arrested Canine Left Ventricle. *Am. J. Physiol.* 261, H918–H928 (1991)
19. Holmes, J.W.: Determinants of Left Ventricular Shape Change during Filling. *J. Biomech. Eng.* 126, 98–103 (2004)
20. Yin, F.C.P., Chan, C.C.H., Judd, R.M.: Compressibility of Perfused Passive Myocardium. *Am. J. Physiol.* 271, H1864–H1870 (1996)

A Hybrid Parallel Algorithm for Transforming Finite Element Functions from Adaptive to Cartesian Grids

Oliver Fortmeier and H. Martin Bücker

RWTH Aachen University
Institute for Scientific Computing
Aachen, Germany
`{fortmeier,buecker}@sc.rwth-aachen.de`

Abstract. High-performance computing is often used in various fields of computational science and engineering where the availability of a large memory is crucial to represent given problems. For problems arising from the discretization of partial differential equations, adaptivity is an alternative approach to reduce the storage requirements. We present an illustrating example demonstrating the need for an algorithm change when a serial hierarchical data structure is parallelized. More precisely, we investigate this issue taking as example an algorithm to transform finite element functions defined on an adaptive grid into functions on a Cartesian grid. By combining distributed- and shared-memory parallelization, the resulting hybrid parallel algorithm involves two levels of parallelism. The performance of the parallel algorithm is shown to be highly problem dependent. There are certain problem instances where the speedup is satisfactory while there is a dramatic performance reduction for other problem instances. The numerical experiments are given for transformations resulting from three-dimensional computational fluid dynamic problems. The problem instances are given by up to 750 000 tetrahedra representing about 3 millions of unknown variables. The experiments are carried out on a Xeon-based cluster using up to 64 processors.

1 Introduction

High-performance scientific computing is intensively used to simulate phenomena occurring in various fields of computational science and engineering. Aircraft simulations, medical, physical and chemical simulations are just a few examples. Combined with real-life experiments, such as wind tunnel experiments or concentration measurements, simulations are increasingly becoming an indispensable tool to gain insight into these phenomena. An interdisciplinary team of researchers at RWTH Aachen University is interested in studying the effects of multi-phase flow problems using high-resolution measurements and advanced numerical simulations [1]. Of particular interest are flow phenomena at the surface of levitated droplets surrounded by liquids [2][3]. This research initiative

aims at advancing mathematical models, measurement techniques, and numerical algorithms. The real-life experiments are carried out by non-invasive nuclear magnetic resonance (NMR) measurements resulting in data on a Cartesian grid. Within this collaboration, the parallel finite element solver DROPS [4] is being developed to simulate two-phase flows, e.g., simulating levitated droplets [5] or falling films [6,7]. This parallel software is capable of adaptively solving the three-dimensional Navier-Stokes equations, specifically designed for two-phase flow problems [8]. Computational subdomains in the vicinity of the interfacial region are discretized by a large number of tetrahedra to obtain a high resolution in this area whereas the remaining subdomains are discretized by using less tetrahedra. This leads to a hierarchy of tetrahedral grids to represent the computational domain. The complicated nature of the underlying two-phase flow problem requires the use of high-performance computers to cope with the high demand of computing time and storage.

In the present situation, the results gained by the parallel simulations and by real-life experiments are available in different data formats. To be able to compare these results, a post processing transformation is necessary, mapping the results of the simulation to the corresponding NMR measurements. To this end, we introduce a hybrid parallel algorithm that differs from a serial algorithm working on an adaptive data structure. The strategy for the distributed-memory parallelization is based on decomposing a hierarchy of tetrahedral grids. It is designed for efficiently solving the two-phase flow problem in parallel. This is by far the most time-consuming part of the overall simulation compared to the transformation. Hence, it is reasonable to not change the parallelization strategy for the transformation. We do not treat the transformation as an separate problem as in [9]. We rather have to consider it as post processing step that is tightly integrated into the given distributed-memory parallelization. Due to this overall strategy, it is not possible to parallelize the given serial algorithm. Therefore, we have to switch from the serial adaptive algorithm to a parallel non-adaptive algorithm, trading off adaptivity for parallelism. Besides the distributed-memory parallelization, an additional level of shared-memory parallelization is introduced leading to a hybrid MPI/OpenMP-parallel algorithm.

The outline of this note is as follows. In Sect. 2 we briefly describe the hierarchy of tetrahedral grids to represent the computational domain. Section 3 deals with the serial and parallel algorithm for transforming solutions on unstructured tetrahedral grids to Cartesian grids. Finally, in Sect. 4 we show performance results dealing with varying problem sizes as well as different numbers of processes and threads.

2 Parallel Hierarchy of Tetrahedral Grids

The main idea of the distributed-memory parallelization is based on a domain decomposition strategy. First, we briefly describe the serial tetrahedral hierarchy which is used by DROPS to discretize the three-dimensional computational domain Ω . To this end, we introduce a hierarchical triangulation and additional

notations. A tetrahedral hierarchy consists of multiple levels of triangulations. The triangulation on level $k = 0$, denoted by \mathcal{T}_0 , defines the coarsest representation of the domain. The refinement algorithm [10] implemented in DROPS generates a sequence of triangulations to obtain a high resolution of the sub-domains of interest. The triangulation on the finest level l is denoted by \mathcal{T}_l . A sequence of triangulations $\mathcal{M} = (\mathcal{T}_0, \dots, \mathcal{T}_l)$ is called a multi-level triangulation if the two following conditions are satisfied for all $k \in \{1, \dots, l\}$:

1. A tetrahedron $T \in \mathcal{T}_k$ is either in \mathcal{T}_{k-1} or it is obtained from a refinement of a tetrahedron in \mathcal{T}_{k-1} .
2. If a tetrahedron $T \in \mathcal{T}_{k-1}$ is not refined when going from \mathcal{T}_{k-1} to \mathcal{T}_k , then T remains unrefined in all triangulations on levels k, \dots, l .

Thus, we can assign each tetrahedron a unique level $k \in \{1, \dots, l\}$ on which it occurs the first time. The set \mathcal{G}_k of all tetrahedra of a level k is called hierarchical surplus. The hierarchical decomposition $\mathcal{H} = (\mathcal{G}_0, \dots, \mathcal{G}_l)$ is the sequence of these surpluses. In Fig. 1, an example of a tetrahedral hierarchy of four levels is illustrated. For the sake of simplicity, this figure shows a triangulation in two space dimensions where a triangle corresponds to a tetrahedron. In Fig. 1(a), a hierarchical decomposition $\mathcal{H} = (\mathcal{G}_0, \dots, \mathcal{G}_3)$ is presented, and in Fig. 1(b) the corresponding multi-level triangulation $\mathcal{M} = (\mathcal{T}_0, \dots, \mathcal{T}_3)$ is shown. In this example, three refinements are performed. Each of these refinements is carried out at the triangle located at the bottom right corner of the unit square. DROPS stores the hierarchical decomposition \mathcal{H} rather than the multi-level triangulation \mathcal{M} in order to store each tetrahedron only once. An admissible hierarchical decomposition of the tetrahedral hierarchy used in the parallel implementation of DROPS was developed by Groß et al. [11]. A crucial point of this decomposition is the

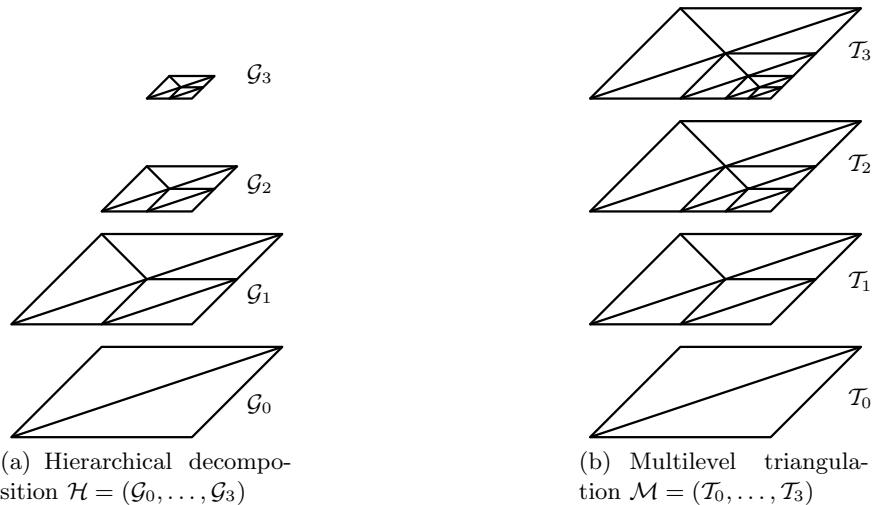


Fig. 1. Hierarchy of four triangulations

vertical access between tetrahedra: If a so-called parent tetrahedron T is refined in at most eight child tetrahedra T_1, \dots, T_k ($2 \leq k \leq 8$) the tetrahedron T needs access to all its children and, vice versa, all children need access to their parent. Using a non-overlapping strategy to store the tetrahedra by multiple processes would imply that complete tetrahedron families are stored by a single process. This may lead to a coarse granularity if a small subdomain of the computational domain Ω is refined multiple times. Therefore, copies of tetrahedra, so-called ghost tetrahedra, are introduced. Assume process p stores T and another process q stores all children T_1, \dots, T_k of T . Then p stores the master copy of T and q stores a ghost copy. This ensures the vertical access between T and its children T_1, \dots, T_k on process q . It is important for this note that all master copies of tetrahedra have access to their parent tetrahedra. However, ghost tetrahedra cannot access their parents because they may not exist on the same process. To balance the computational work among the processes, we uniformly distribute the tetrahedra of the finest triangulation \mathcal{T}_l among the processes. This leads to local triangulations \mathcal{T}_k^p of level k on each process p . For a detailed description of the admissible hierarchical decomposition, we refer to [11].

3 Transformation to Cartesian Grids

After having introduced triangulations and the decomposition of triangulations used by DROPS in the previous section, we outline the concept of transforming a finite element function from a triangulation to a Cartesian grid in Sect. 3.1. Afterward, we describe a fast serial algorithm in Sect. 3.2. We present a distributed-memory parallelized in Sect. 3.3 and a hybrid parallel algorithm in Sect. 3.4.

3.1 Concept of Transformation

Without loss of generality, we restrict the finite element functions to the finest level of the triangulation \mathcal{T}_l . The finite element functions on \mathcal{T}_l used by DROPS are either linear or quadratic. Let φ be a linear finite element function, then this function is uniquely defined by the values on vertices in \mathcal{T}_l . If φ is a quadratic function then the values on vertices and edges are needed to define this function. Determining the value of φ within a single tetrahedron $T \in \mathcal{T}_l$ can be done by taking only local data on T . We denote the local interpolation of a finite element function φ on a single tetrahedron T by $I_T(\varphi)$.

In this note, we describe a Cartesian grid \mathcal{R} by specifying the number of grid points $n_x \times n_y \times n_z$ in each spatial direction. In order to transform a finite element function φ described on \mathcal{T}_l to a Cartesian grid \mathcal{R} , we have to determine the values of φ on each grid point $\mathbf{q} \in \mathbb{R}^3$. Figure 2 depicts a high-level description of this transformation. This algorithm takes as input a finite element function φ and a triangulation \mathcal{T}_l and produces a three-dimensional tensor R of the interpolated values on the Cartesian grid. Therefore, for each grid point \mathbf{q} in \mathcal{R} , a tetrahedron \widehat{T} is determined which surrounds \mathbf{q} . Afterward, the local interpolation $I_{\widehat{T}}(\varphi)$ is applied to obtain the value of φ on \mathbf{q} .

On the one hand, this algorithm determines \mathbf{q} by an affine mapping defined by the parameters of \mathcal{R} . On the other hand, interpolating a finite element function on a single tetrahedron only involves local data. Denoting the number of tetrahedra in triangulation \mathcal{T}_l by $n_{\mathcal{T}_l}$, line 4 and line 6 are constant in time with respect to $n_{\mathcal{T}_l}$. However, finding the tetrahedron $\widehat{T} \in \mathcal{T}_l$ containing \mathbf{q} obviously depends on the number of tetrahedra. Consequently, the main computational work of Fig. 2 is spent in line 5.

```

1: for  $i = 1$  to  $n_x$  do
2:   for  $j = 1$  to  $n_y$  do
3:     for  $k = 1$  to  $n_z$  do
4:       Determine  $\mathbf{q} \in \mathbb{R}^3$  described by  $i, j$  and  $k$ 
5:       Find  $\widehat{T} \in \mathcal{T}_l$  containing  $\mathbf{q}$ 
6:        $R(i, j, k) = I_{\widehat{T}}(\varphi)(\mathbf{q})$ 
7:     end for
8:   end for
9: end for
```

Fig. 2. Basic algorithm for transforming a finite element function φ described on a triangulation \mathcal{T}_l to a Cartesian grid

3.2 Serial Algorithm

Next, we outline a fast serial algorithm for finding \widehat{T} depicted in Fig. 3. Recall that a hierarchy of the triangulations is stored. Therefore, a hierarchical approach can be applied to find the tetrahedron \widehat{T} which surrounds the point \mathbf{q} . In the first phase of this algorithm, we iterate over all tetrahedra of the coarsest triangulation \mathcal{T}_0 until we find a tetrahedron T_0 containing \mathbf{q} . If we do not find such a tetrahedron T_0 corresponding to \mathbf{q} then \mathbf{q} is located outside of the computational domain Ω . Here, we set $I_{T_0}(\phi) := 0$. Otherwise, let $T_0 \in \mathcal{T}_0$ denote the tetrahedron which contains \mathbf{q} . In the second phase of the serial algorithm, we search within the family of T_0 for a descendant \widehat{T} in the finest triangulation \mathcal{T}_l containing \mathbf{q} .

3.3 Parallel Algorithm

The serial algorithm for finding $\widehat{T} \in \mathcal{T}_l$ given in Fig. 3 is not applicable for the parallel decomposition of the tetrahedral hierarchy, because we do not store complete tetrahedral families by one process. Therefore, we cannot exploit the hierarchical structure of the triangulations in the same way as the serial algorithm does. Recall from Sect. 2 that the finest triangulation \mathcal{T}_l is uniformly distributed among all processes, leading to \mathcal{T}_l^p on process p . Therefore, we can independently search for the tetrahedron $\widehat{T} \in \mathcal{T}_l^p$ containing \mathbf{q} by each process p . Each tetrahedron has access to its parent tetrahedron, either as ghost or as master tetrahedra. Hence we can improve the parallel algorithm. Instead of searching for \widehat{T} in the finest triangulation \mathcal{T}_l^p , each process searches for T_0 in

```

1: Find  $T_0 \in \mathcal{T}_0$  containing  $\mathbf{q}$ 
2: if no  $T_0$  found then
3:   return  $\emptyset$ 
4: else
5:    $\widehat{T} \leftarrow T_0$ 
6:   while  $\widehat{T}$  is refined do
7:     Find  $\bar{T}$  as a child of  $\widehat{T}$  containing  $\mathbf{q}$ 
8:      $\widehat{T} \leftarrow \bar{T}$ 
9:   end while
10:  return  $\widehat{T}$ 
11: end if

```

Fig. 3. Sequential algorithm for finding a tetrahedron \widehat{T} in the finest triangulation \mathcal{T}_l , which contains a given point \mathbf{q}

the triangulation \mathcal{T}_{l-1}^P . If T_0 is refined, then we determine a child tetrahedron $\widehat{T} \in \mathcal{T}_l^P$ of T_0 that contains the point \mathbf{q} . Otherwise, T_0 is the requested tetrahedron. Since each process stores a local triangulation \mathcal{T}_l^P corresponding to a part of the computational domain, this implies that each process fills only parts of R with the interpolated values of φ . These parts are determined by the intersection of \mathcal{R} and the tetrahedra of \mathcal{T}_l^P . If R is needed on one or on all processes, an additional reduction operation is necessary to obtain R . We summarize the parallel algorithm in Fig. 4.

```

1: for  $i = 1$  to  $n_x$  do
2:   for  $j = 1$  to  $n_y$  do
3:     for  $k = 1$  to  $n_z$  do
4:       Determine  $\mathbf{q} \in \mathbb{R}^3$  described by  $i, j$  and  $k$ 
5:       Find  $T_0 \in \mathcal{T}_{l-1}^P$  containing  $\mathbf{q}$ 
6:       if  $T_0$  exists then
7:         if  $T_0$  is refined then
8:            $\widehat{T} \leftarrow$  child of  $T_0$  containing  $\mathbf{q}$ 
9:         else
10:           $\widehat{T} \leftarrow T_0$ 
11:        end if
12:         $R(i, j, k) = I_{\widehat{T}}(\varphi)(\mathbf{q})$ 
13:      else
14:         $R(i, j, k) = 0$ 
15:      end if
16:    end for
17:  end for
18: end for
19: (Reduce  $R$  to all processes)

```

Fig. 4. Parallel algorithm, executed by each process p , for transforming a finite element function φ described on a distributed tetrahedral hierarchy to a Cartesian grid \mathcal{R}

3.4 Hybrid Parallel Algorithm

Taking a closer look at the basic algorithm in Fig. 2, we observe that all values in \mathcal{R} can be independently computed, i.e., there are no data dependencies within the body of the three nested **for**-loops. Hence, we can parallelize the outermost loop in line 11 of Fig. 4. Since line 5 needs to access the local triangulation \mathcal{T}_{l-1}^p , this additional level of parallelism is utilized with threads. Note that this shared-memory parallelization is also possible for the serial algorithm.

4 Numerical Results

In order to give some performance results of the serial and hybrid parallel algorithm developed in Sect. 3, tests are performed on a cluster of the Center for Computation and Communication of RWTH Aachen University. Each processor of this cluster consists of two Xeon-based quad-core chips which have access to a shared memory. The processors are connected by an InfiniBand network. We are using MPI [12] for the communication among and synchronization of the processes. OpenMP [13] handles the shared-memory parallelization. By using this hybrid parallelization strategy, we exploit the structure of the cluster architecture. We use the notation $P \times T$ to denote a hybrid parallelization consisting of P MPI processes each with T threads. Previous performance tests for solving two-phase flow problems with DROPS indicated a good performance if, on each quad-core chip, a single MPI process is located. Hence, within a parallelization using $T = 1, 2$ or 4 threads per MPI process, each MPI process is bound to a chip. However, if $T = 8$ threads are used per MPI process, an MPI process is assigned to each processor.

Since the software package DROPS is developed for two-phase flows, we consider problems with a spherical oil droplet surrounded by water as illustrating examples. Here, the unit cube and the canonical NMR-measurement cell represent the computational domain. The tetrahedral grid is refined locally in the region near the surface of the droplet. Table 1 summarizes the setup of five problems (A–E), where, as above, we denote the number of tetrahedra in a triangulation \mathcal{T}_k by $n_{\mathcal{T}_k}$. Furthermore, the number of tetrahedra in a surplus \mathcal{G}_k is denoted by $n_{\mathcal{G}_k}$ and the number of unknowns to discretize velocity on Ω is given by n_v .

Table 1. Setup and size of all problems

Prob.	Shape	#level	$n_{\mathcal{T}_0}$	$n_{\mathcal{T}_l}$	$n_{\mathcal{G}_l}$	\mathcal{R}	n_v
A	cube	0	750 000	750 000	750 000	$70 \times 70 \times 70 = 343\,000$	2 940 300
B	cube	1	162 000	411 030	290 058	$70 \times 70 \times 70 = 343\,000$	1 624 074
C	cube	4	384	364 008	340 896	$70 \times 70 \times 70 = 343\,000$	1 456 104
D	conical cell	2	4 635	8 447	3 156	$128 \times 100 \times 37 = 473\,600$	28 812
E	conical cell	4	4 635	103 766	87 127	$128 \times 100 \times 37 = 473\,600$	410 262

Problems A–C are defined on a unit cube Ω with different discretizations. The Cartesian grid for these three problems consists of $70 \times 70 \times 70$ points. These examples vary the discretization of the cube by varying the number of tetrahedra of the coarsest triangulation and the number of levels in the tetrahedral hierarchy. That is, different numbers of refinements are applied to the triangulation close to the interfacial region of the droplet. Problems D and E represent the real-life conical measurement cell. In Fig. 5, the triangulations of the finest level are illustrated by two-dimensional slices.

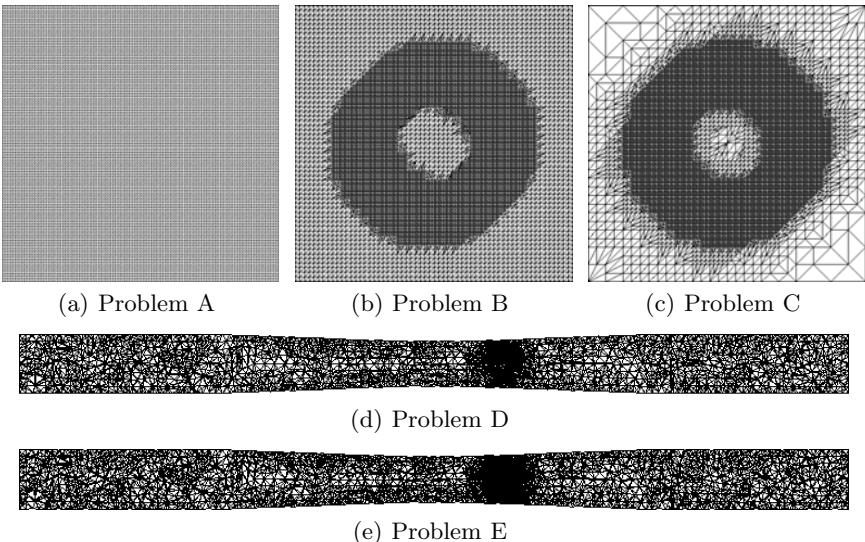


Fig. 5. Two-dimensional slices through the discretized domains

Before concentrating on the parallel performance, we shortly discuss two effects that, in addition to communication and synchronization, account for reducing the ideal speedup:

1. The distribution of data to MPI processes is based on a distribution of the computational work of the two-phase flow problem. Therefore, the finest level of the tetrahedral hierarchy, \mathcal{T}_l , is partitioned. However, for the transformation considered in this note, the distribution of \mathcal{T}_{l-1} would be more appropriate. Thus, the heuristics to distribute the tetrahedra among the processes lead to some load imbalances.
2. There is another reason for further load imbalances. Searching for the tetrahedron $\hat{T} \in \mathcal{T}_l$ is implemented as a linear search in \mathcal{T}_{l-1} . Thus, it may happen that there are some processes that do not store any tetrahedron containing a point in \mathcal{R} and, thus, will never find any \hat{T} . This is an extreme case illustrating that the runtime of the searches may vary significantly, causing load imbalances.

In problem A, the cube Ω is discretized by an equidistant $50 \times 50 \times 50$ grid, where each subcube is represented by 6 tetrahedra. That is, the hierarchy of triangulations only consists of one triangulation with 750 000 tetrahedra. This example is designed to illustrate the behavior of the algorithms without adaptivity. Let t_{ser} be the runtime of the serial algorithm and $t_{P \times T}$ the runtime of the parallel algorithm executed by P processes and T threads per process. The speedup is defined by

$$S_{P \times T} = \frac{t_{\text{ser}}}{t_{P \times T}}.$$

Note, that in general $S_{1 \times 1} < 1$ holds. In Fig. 6(a) the speedup of transforming the velocity is shown. Here, we observe that the distributed-memory parallelization is well suited for this problem. The speedup is increasing with increasing number of MPI processes. For 64 MPI processes, the serial runtime of $t_{\text{ser}} = 1295.2$ s is reduced by a factor of about 45. Applying one local refinement to this triangulation is infeasible on a single processor because the resulting memory needed to store the data representing a two-phase flow problem would exceed the available memory resources.

In problem B, the cube Ω is firstly discretized by an equidistant $30 \times 30 \times 30$ grid, where each subcube is represented by 6 tetrahedra, i.e., the triangulation on level 0 consists of 162 000 tetrahedra. Then, the tetrahedra close to the drop surface are refined. This leads to a total amount of 411 030 tetrahedra in the finest triangulation. Thus, the serial and parallel algorithms search for \hat{T} in the same triangulation, i.e., in the coarsest triangulation \mathcal{T}_0 . Here, the serial runtime of $t_{\text{ser}} = 11603.9$ s is reduced to $t_{2 \times 1} = 9819.8$ s by transforming the finite element function with 2 MPI processes and 1 thread. With 64 MPI processes the serial runtime is reduced further to $t_{64 \times 1} = 570.0$ s.

In contrast to the previous problems where the speedup for the distributed-memory parallelization is reasonable, problem C is designed for the serial algorithm exploiting the hierarchical data structures. Here, the coarsest triangulation \mathcal{T}_0 consists of 384 tetrahedra and the finest triangulation \mathcal{T}_4 of 364 008 tetrahedra. The numbers of tetrahedra in all triangulations and surpluses are printed in Table 2. The serial algorithm transforms the finite element function from the hierarchy of tetrahedral grids to a Cartesian grid in about $t_{\text{ser}} = 0.4$ s. However, using the parallel algorithm reduces the computational time of $t_{2 \times 1} = 73.4$ s on 2 processes by a factor of 29.4 to $t_{64 \times 1} = 2.5$ s on 64 processes (Fig. 6(c)). Using a hybrid approach with $T = 8$ threads per process and $P = 64$ processes reduces the runtime to $t_{64 \times 8} = 0.9$ s. Overall, the parallel algorithm executed on 64 processors needs 0.5 s more than the serial algorithm. Note that the parallel algorithm using the 1×1 approach which does not exploit the hierarchy in the triangulations needs $t_{1 \times 1} = 132.7$ s for the transformation. As expected, all speedups are extremely small for problem C.

Problems D and E are taken from a real-life experiment, where an oil droplet surrounded by water is measured by NMR in a conical measurement cell. These

Table 2. Number of master tetrahedra on hierarchical surpluses and triangulations for problem C

Level k	tetrahedra in \mathcal{T}_k	tetrahedra in \mathcal{G}_k
0	384	384
1	2 784	2 784
2	15 612	15 084
3	71 160	65 388
4	364 008	340 896

measurements are taken with a resolution of $128 \times 100 \times 37$ in \mathcal{R} . In Fig. 6(d)–Fig. 6(e) the speedup for problem D and E with respect to different numbers of MPI processes and threads are shown.

Problem D is the smallest problem in terms of number of tetrahedra in the finest triangulation. The serial algorithm takes $t_{\text{ser}} = 446.8$ s to transform the finite element function on the tetrahedral grid into a function on the Cartesian grid described by 473 600 points. Since the hierarchy of triangulations just consists of two levels the distributed-memory parallelization works as expected and leads to a speedup of $S_{64 \times 1} = 29.1$.

The serial algorithm needs approximately the same time for transforming the finite element function in problem E and D, i.e., $t_{\text{ser}} = 446.8$ s and $t_{\text{ser}} = 446.9$ s, respectively. However, in contrast to problem D, 8 MPI processes still consume more runtime for the transformation as the serial algorithm. But with an increasing number of processes the runtime is reduced below the serial runtime. Here, we reach a speedup of $S_{64 \times 1} = 6.4$. Again, an additional benefit comes from the hybrid parallel approach, e.g., a speedup of $S_{32 \times 4} = 7.7$ is observed.

In Fig. 7 we focus on the speedup of the shared-memory parallelization for all five examples. More precisely, the speedup of the OpenMP-parallelized serial algorithm using T threads and the parallel algorithm using a hybrid $P \times T$ strategy with $P = 4, 16$ and $P = 64$ is depicted when varying the number of threads per process $T \in \{1, 2, 4, 8\}$. The use of a different number of MPI processes leads to similar results as in Fig. 7(b)–Fig. 7(d) and are omitted. Since the runtime of the linear search of the tetrahedron \hat{T} differs, we use the OpenMP dynamic scheduling. Searching for the tetrahedron \hat{T} is dominated by memory bandwidth. Thus, using 2 threads can exploit the architecture of the Xeon-based processors well, i.e., this parallelization leads to a speedup of about 1.75 for all problems, except for the OpenMP-parallelized algorithm applied to problem C. Here, the OpenMP-parallelized serial algorithm just leads to an efficiency of 70 %. Using 4 threads reduces the runtime by a factor of about 2 for the serial and the distributed-memory parallel algorithm. The underlying MPI parallelization with 8 threads differs from the other parallelization strategies. Here, each processor is just responsible for one MPI process in contrast to 2 processes per processor. This strategy leads to an efficiency of less than 50 % for most cases. For all problems, the efficiency of the shared-memory parallelization is slightly reduced when increasing the number of MPI processes. Overall the shared-memory parallelization gives a benefit for all problems and for all number of MPI processes.

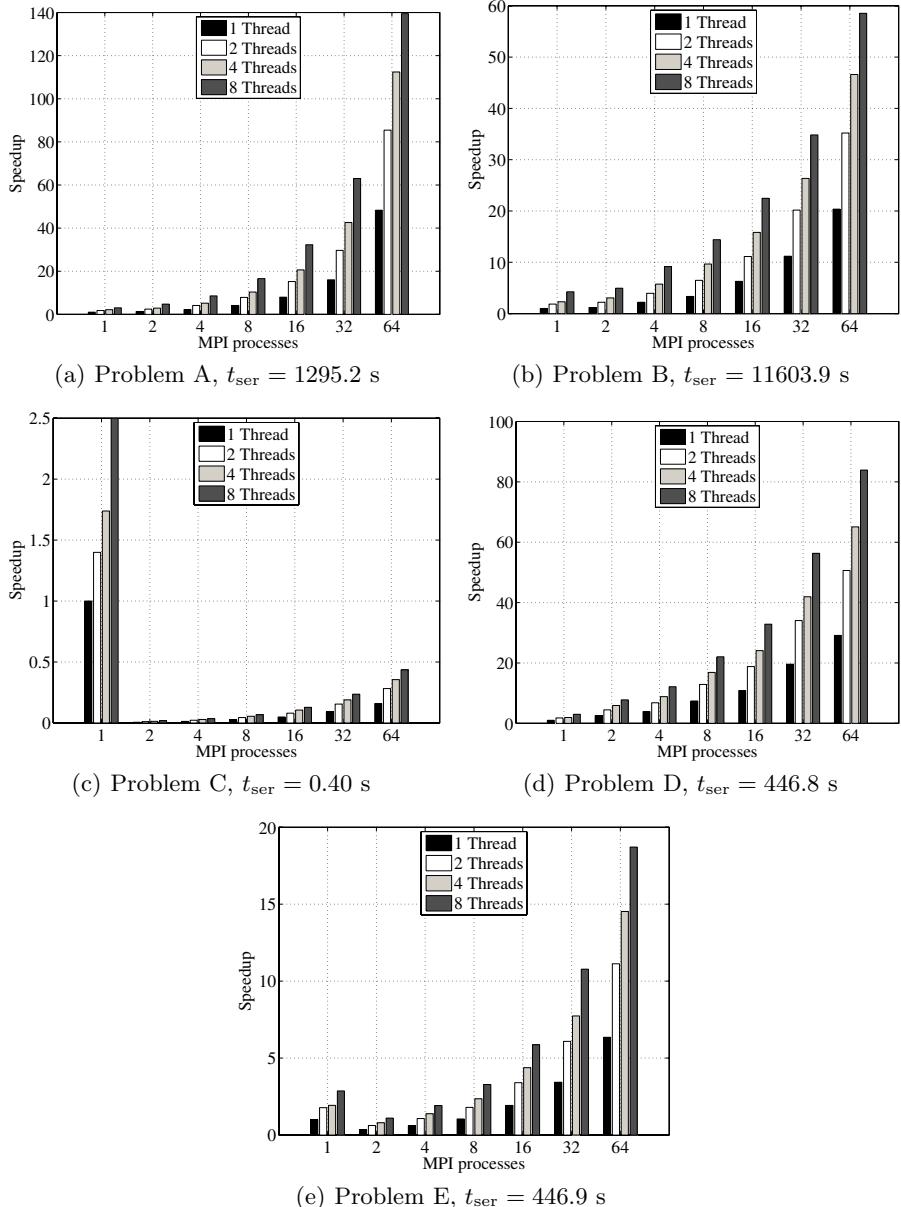


Fig. 6. Speedup of problems A–E with respect to the serial algorithm which is optimized for the hierarchical data structure

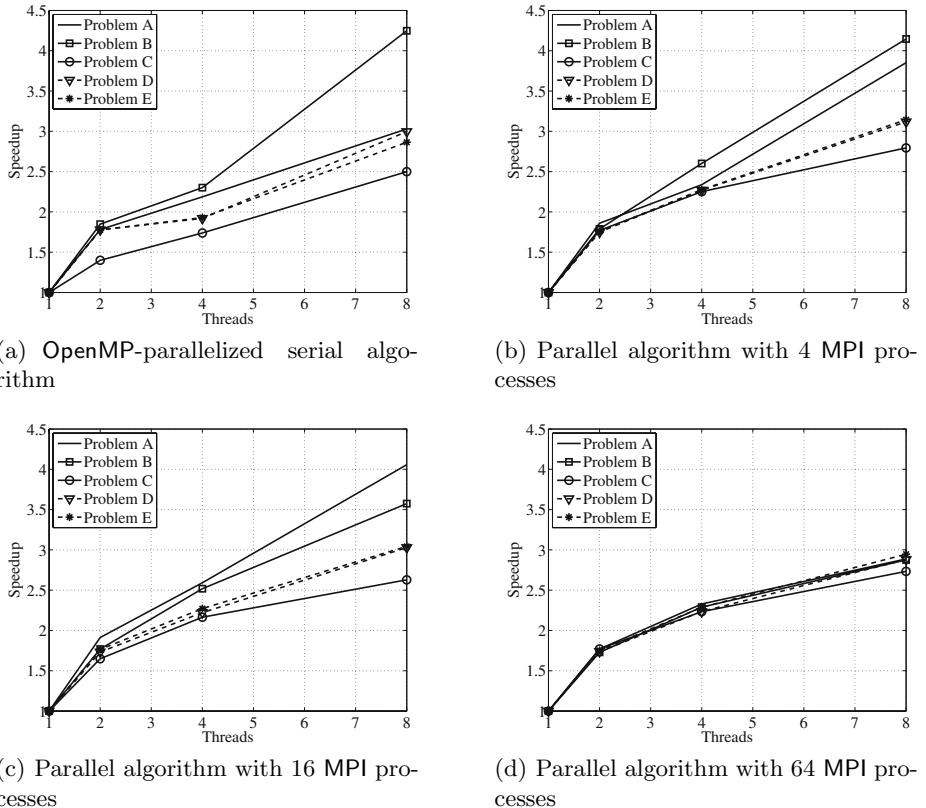


Fig. 7. Speedup of the shared-memory parallelization

5 Conclusion

Recent finite element solvers are often using a distributed-memory parallelization for solving or even representing problems occurring in various fields of science and engineering. While switching from a serial data structure to a distributed data structure, various algorithms within these finite element solvers have to be modified or even replaced. Here, we consider the parallelization of a post processing step in the context of a larger MPI-parallelized simulation. More precisely, an algorithm for transforming finite element functions from adaptive to Cartesian grids is parallelized. The serial algorithm is optimized for a serial data structure describing a hierarchy of triangulations. The new parallel algorithm has a higher complexity than the serial one. However, the parallel runtime is, in most cases, smaller than the serial runtime if the number of processors is large enough.

The parallelization strategy employs two levels of parallelism. The first level stems from distributing the tetrahedral hierarchy to different MPI processes. A second level is introduced by distributing the Cartesian grid to different threads. Due to the structure of the serial and parallel algorithm, OpenMP turns out

to be a suitable programming model for shared-memory parallelization of the second level. By using the hybrid parallel approach, i.e., a distributed/shared-memory parallelization, the algorithm exploits the architecture of a Xeon-based multi-core cluster.

There is room for further performance improvements. For instance, higher resolution requirements increase the number of tetrahedra on the finest triangulation. This might lead to problem instances for which the new parallel algorithm is not well suited because the values on the Cartesian grid points are determined one after another by searching for the proper tetrahedra. Therefore, future research activities will be directed toward improving the performance by exploiting the geometric structure to find the proper tetrahedra. Another direction may also be advantageous, namely, rearranging the algorithm to iterate over the tetrahedra and assigning values to the proper grid points corresponding to the current tetrahedron.

Acknowledgment

The development of the parallel finite element solver DROPS is done in a collaborative work with the Chair of Numerical Mathematics, RWTH Aachen University, Germany. This research is supported by the Deutsche Forschungsgemeinschaft (DFG) within SFB 540 “Model-based experimental analysis of kinetic phenomena in fluid multi-phase reactive systems”, RWTH Aachen University and by the German Federal Ministry of Education and Research (BMBF) under the contract 03SF0326A “MeProRisk: Novel methods for exploration, development, and exploitation of geothermal reservoirs - a toolbox for prognosis and risk assessment.” The authors would also like to thank the Center for Computation and Communication of RWTH Aachen University for their support performing the performance tests.

References

1. Marquardt, W.: Model-based experimental analysis of kinetic phenomena in multi-phase reactive systems. *Transactions of the Institution of Chemical Engineers* 83(A6), 561–573 (2005)
2. Gross-Hardt, E., Amar, A., Stapf, S., Pfennig, A., Blümich, B.: Flow dynamics inside a single levitated droplet. *Industrial & Engineering Chemical Research* 1, 416–423 (2006)
3. Gross-Hardt, E., Slusanschi, E., Bücker, H.M., Pfennig, A., Bischof, C.H.: Practical shape optimization of a levitation device for single droplets. *Optimization and Engineering* 9(2), 179–199 (2008)
4. Groß, S., Reichelt, V., Reusken, A.: A finite element based level set method for two-phase incompressible flows. *Comput. Vis. Sci.* 9(4), 239–257 (2006)
5. Bertakis, E., Groß, S., Grande, J., Fortmeier, O., Reusken, A., Pfennig, A.: Validated simulation of droplet sedimentation with finite-element and level-set methods. *Comp. Eng. Sci.* (2008)

6. Groß, S., Soemers, M., Mhamdi, A., Al-Sibai, F., Reusken, A., Marquardt, W., Renz, U.: Identification of boundary heat fluxes in a falling film experiment using high resolution temperature measurements. *International Journal of Heat and Mass Transfer* 48, 5549–5562 (2005)
7. Bücker, H.M., Willkomm, J., Groß, S., Fortmeier, O.: Discrete and continuous adjoint approaches to estimate boundary heat fluxes in falling films. *Optimization Methods and Software* (2008)
8. Groß, S., Reusken, A.: Finite element discretization error analysis of a surface tension force in two-phase incompressible flows. *SIAM J. Numer. Anal.* 45, 1679–1700 (2007)
9. Wolter, M., Schirski, M., Kuhlen, T.: Hybrid Parallelization for Interactive Exploration in Virtual Environments. In: *Parallel Computing: Architectures, Algorithms and Applications. Advances in Parallel Computing*, vol. 15, pp. 79–86. IOS Press, Amsterdam (2007)
10. Bey, J.: Simplicial grid refinement: On Freudenthal's algorithm and the optimal number of congruence classes. *J. Numer. Math.* 85(1), 1–29 (2000)
11. Groß, S., Reusken, A.: Parallel multilevel tetrahedral grid refinement. *SIAM J. Sci. Comput.* 26(4), 1261–1288 (2005)
12. Walker, D.W., Dongarra, J.J.: MPI: A standard Message Passing Interface. *Supercomputer* 12(1), 56–68 (1996)
13. OpenMP Architecture Review Board: OpenMP Application Program Interface 3.0, Version 3.0 (May 2008)

Leveraging Multicore Cluster Nodes by Adding OpenMP to Flow Solvers Parallelized with MPI

Christian Iwinsky, Samuel Sarholz, Dieter an Mey, and Ralph Altenfeld

JARA, RWTH Aachen University, Center for Computing and Communication

Seffenter Weg 23, 52074 Aachen, Germany

{iwinsky,sarholz,anmey,altenfeld}@rz.rwth-aachen.de

Abstract. MPI is the predominant model for parallel programming in technical high performance computing. With an increasing number of cores and threads in cluster nodes the question arises whether pure MPI is an appropriate approach to utilize today's compute clusters or if it is profitable to add another layer of parallelism within the nodes by applying OpenMP on a lower level. Investing a limited amount of manpower, we add OpenMP directives to three MPI production codes and compare and analyze the performance varying the number of MPI processes per node and the number of OpenMP threads per MPI process on current CMP/CMT architectures.

1 Introduction

Today's common HPC platforms are clusters with multicore nodes linked together with a fast interconnect and typically scientific applications employ message passing with MPI to utilize such a resource. In order to increase efficiency, multiple MPI processes are scheduled per node, frequently as many as there are cores available. Furthermore, memory bandwidth, network-bandwidth and -latency and cache utilization suffer from overloading the single nodes, thus preventing an efficient usage of multicore technology. There is no indication that network latency can be reduced at the same rate as the number of processor cores are predicted to grow. A potential approach to alleviate this problem is to exploit the local shared memory properties of modern multicore CPUs.

Here we investigate how adding OpenMP [1], the de-facto standard for thread level parallelization in scientific applications, to existing MPI codes without numerical or large algorithmic changes can influence scalability. The evaluation targets for this work are three flow simulation codes (XNS [2], FS3D [3], CD-PLit [4],[5]) currently developed and used at the RWTH Aachen University. These codes originally only use MPI for parallel computing.

This paper is organized as follows: In Section 2 and 3 we briefly describe the flow solvers and the test environment and in Section 4 we present the results of the MPI scalability study. Section 5 then describes our efforts to add OpenMP parallelization, and the resulting changes in runtime behavior are presented in Section 6. Finally we draw our conclusion and follow up with potential future work in section 7.

2 The Codes

For this work we investigate the following flow solver codes (see table ②):

XNS ② is a CFD solver based on finite elements written primarily in Fortran and a little C developed by the chair for Computational Analysis of Technical Systems of the RWTH Aachen University. The code has been successfully adapted to several platforms. The data set used for this test is the simulation of the Ohio Dam spill way.

FS3D ③ is a direct numerical flow solver for two phase flows written in Fortran. Initial development started at the Institute of Aerospace Thermodynamics of the University of Stuttgart and continued amongst others by Univ.-Prof. Dr. Dieter Bothe. The code is based on a volume-of-fluid (VOF) method and solves the Navier-Stokes equation for incompressible flows taking energy transport and phase transition into account.

CDPLit ④ is a direct numerical solver for two phase flows written in Fortran and C developed at the Institute for Combustion Technology of the RWTH Aachen University. The code is based on finite volumes operators using refined level set grids. CDPLit combines the flow solver CDP with the LIT level set by the means of the coupling software CHIMPS. The data set used for this work is the simulation of a diesel injection spray. Initially all the codes run in their default configuration for many thousands of iterations. To keep runtime and later measurements times down to a minimum, each data set was modified to run for only a few iterations.

3 The Test Systems

For our tests we used the Intel Xeon based cluster of the RWTH Aachen University. This cluster consists of 266 nodes with 2 quadcore Intel Xeon Harpertown (RX200) CPUs running at 3GHz. Each node has 16GB of RAM, a DDR InfiniBand interconnect, local scratch storage as well as a system wide network file system. The operating system is Linux CentOS version 5.2. We compiled with version 11.0 of the 64 bit Intel compilers and linked to the Sun MPI library, version 8.1. The Sun Studio compiler was also used in the OpenMP parallelization efforts for its autoscoping ⑥ capability. For performance measurements, the Sun Analyzer for x86 (version 7.7) and Vampir (version 5.4.4) were used.

4 MPI Scalability

We performed scalability measurements for each of the codes with 1, 2, 4 and 8 processes per node (PPN) for 1, 2, 4, 8, 16, 32, 64, 128 nodes to evaluate which setting would provide the best performance. We also ensured that the processes were scheduled to the compute nodes exclusively. Each measurement was repeated multiple times, depending on the variation of the timing results. Figures ①a, ①b and ①c show the results of these measurements for XNS, FS3D and CDPLit respectively.

Table 1. MPI only, best effort

	XNS			FS3D			CDPLit		
	RT	NPROC	NN	RT	NPROC	NN	RT	NPROC	NN
1 PPN	63.79 s	64	64	17.56 s	128	128	12.67 s	64	64
2 PPN	79.27 s	64	32	16.64 s	256	128	11.48 s	128	64
4 PPN	122.63 s	32	8	30.72 s	128	32	13.28 s	128	32
8 PPN	260.60 s	32	4	76.04 s	128	16	22.58 s	128	16

RT=runtime; NPROC=number of MPI processes;

NN=number of nodes; PPN=processes per node

For XNS the peak performance was at 64 nodes with 64 MPI processes. The other scheduling schemes had their best runtimes in the area of 32 to 64 MPI processes. Table 1 provides more details. With FS3D, the best runtime were obtained at 128 nodes with 256 processes. Again, the fastest runtime for the other scheduling schemes was in the same area. With CDPLit, we saw a similar pattern, obtaining the best runtime with 64 nodes and 128 processes. The other scheduling schemes reached their best runtime around the same MPI process count.

In general, we observed that filling a node with 8 MPI processes scales very well for up to 4 nodes, but with more nodes runtime increases again. Scheduling 4 processes per node is typically as good as 8 processes per node but continues to scale much further. This indicates that for up to 16 nodes, it is most efficient to schedule 4 processes per node. However, to gain the most speedup, one has to deploy much more hardware and schedule only 1 or 2 processes per node.

In addition, we observed a considerable variance between sets of measurements. For XNS, the largest difference from the fastest run to the slowest in a set was 5.6% and 2% on average. For CDPLit, this difference had a maximum of 12.25% with an average of 3%, and for FS3D, the maximum difference was 23.7% at an average of 6.5%. Lacking further explanation, we attributed this variance to the accumulated effect of OS-jitter.

5 OpenMP Implementation

After observing these results, we attempted to improve the utilization of the available cores, by adding OpenMP to the aforementioned codes. The goal for this was to improve scalability without modifying the numerical algorithms and by investing only a limited amount of manpower.

In the first step, the hot spots, that is to say compute intensive regions, were located using the Sun Performance Analyzer and Vampir. For this, we ran each program serially on a single, empty node. With the information from the analyses and a brief study of the code these hot-spots, typically nests of loops, were parallelized using OpenMP constructs. However no in-depth knowledge of the codes and the underlying algorithms was available. As all of the hot-spots

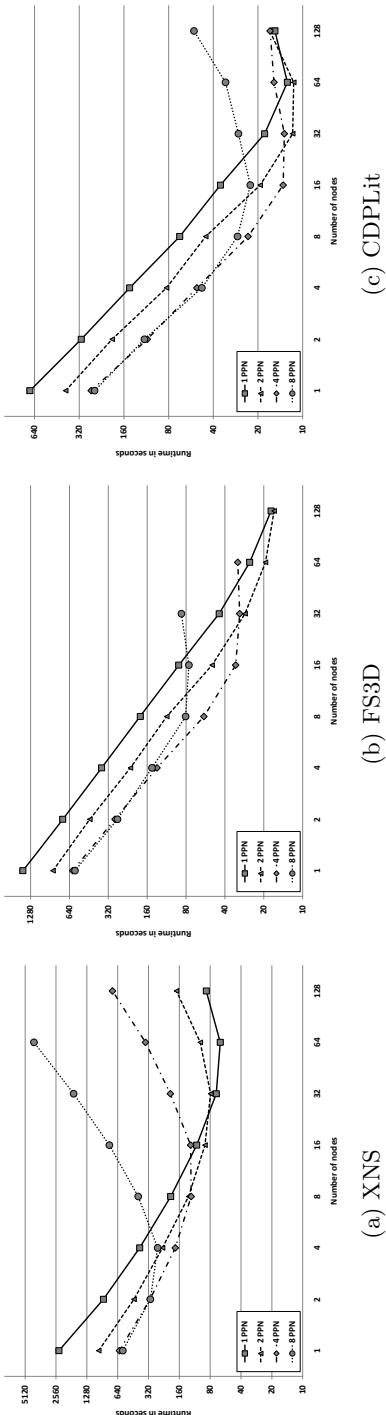


Fig. 1. MPI only scalability, 1 to 128 nodes

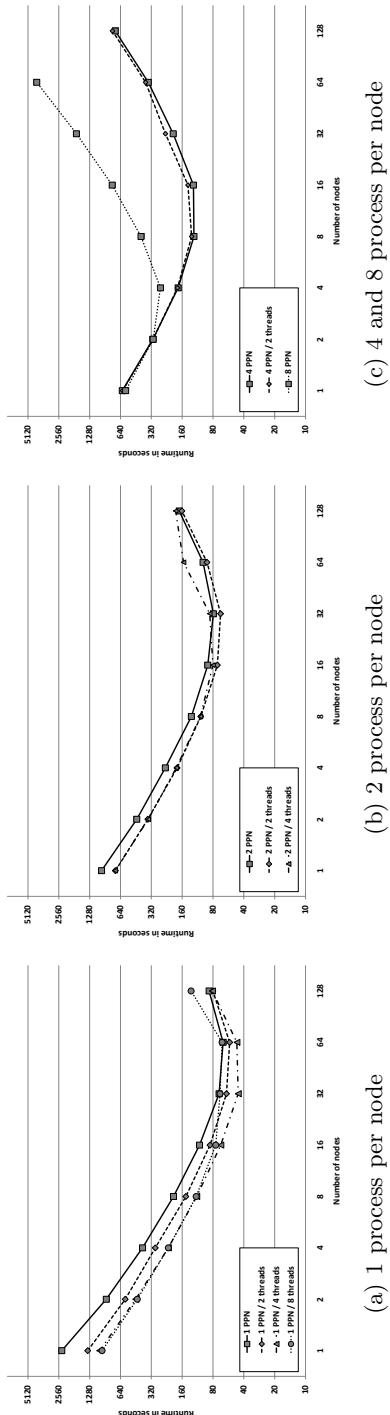


Fig. 2. XNS hybrid scalability, 1 to 128 nodes

were encountered in Fortran loops, OpenMP parallel do-constructs were used to distribute the work amongst threads. In this step, we relied heavily on the autoscoping feature of the Sun Compiler to assist with the variable scoping. For functions in the parallel regions that could not be determined to be thread-safe we added master constructs followed by barriers to ensure parallel correctness. For MPI-communication within the parallel regions, we used the same approach. This corresponds to a hybrid master-only approach [7] in which only the master-thread communicates.

In the second step, the OpenMP parallel regions were extended to reduce the OpenMP overhead. In this step, orphaning – implementing work-sharing constructs in a different lexical scopes than the enclosing parallel construct – was employed to push the OpenMP-parallel directive further up in the call chain.

Finally, all major hot-spots of the three codes were parallelized with OpenMP in addition to the MPI parallelization. An overview of the OpenMP constructs used can be obtained from Table 2. Overall approximately 4 days of work was spent on each code to implement these changes.

Table 2. Code overview and OpenMP constructs used

code	Lines of code	Parallel regions	Number of used constructs
XNS	48k	9	9 do, 2 critcal
FS3D	77k	78	82 do, 3 master, 1 reduction
CDPLit	160k	2	2 do, 7 master, 4 workshare, 6 barrier, 1 reduction

6 Hybrid Scalability

To check for potential better scalability the measurements from section 4 were repeated. However, this time some empty cores were populated with OpenMP threads. If only 1 MPI process was scheduled per node, then measurements were performed with only 1 thread, 2, 4 or 8 threads. For 2 MPI processes per node measurements were performed with 1, 2 or 4 threads per process, and for 4 MPI processes, measurements were performed with 1 and 2 threads.

We did not apply any thread-binding, as the 8.1 release of the Sun MPI does not provide the means to bind the processes and threads automatically to specific cores. However, a few tests with Intel MPI 3.2 indicated that the impact of binding for these tests was negligible, as all the cores of a single node are attached to a common north bridge, thus sharing the same path to main memory.

The resulting scalability measurements can be seen in Figures 2a, 2b and 2c for XNS; 3a, 3b and 3c for FS3D; 4a, 4b and 4c for CDPLit.

Again we see that completely filling the nodes, even with threads, does not achieve the best results for larger node counts. From our observation, we conclude

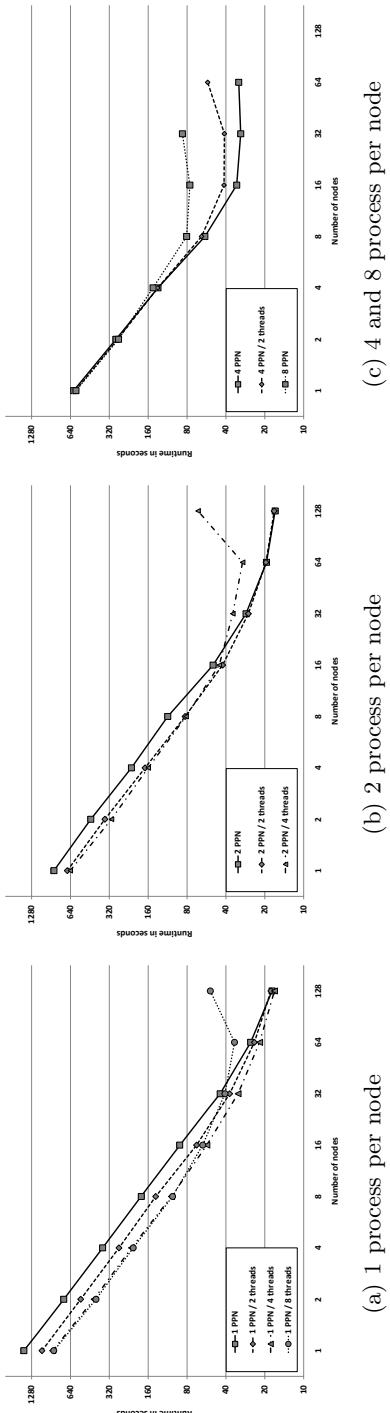


Fig. 3. FS3D hybrid scalability, 1 to 128 nodes

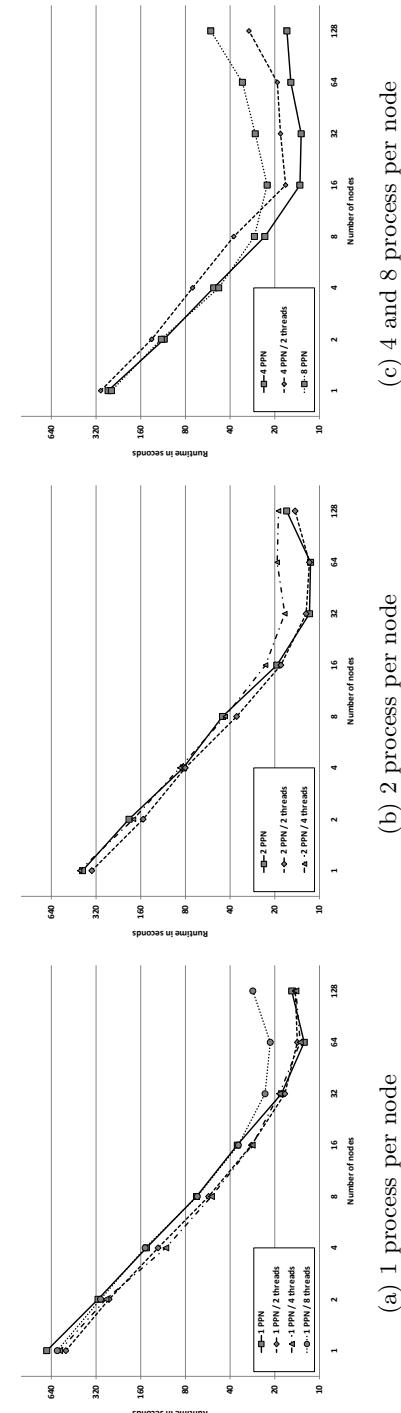


Fig. 4. CDPLit hybrid scalability, 1 to 128 nodes

that typically one can efficiently utilize no more than 4 cores. Adding additional threads, however, does not decrease the performance as much as with adding additional MPI processes to a node.

For XNS, we see that the hybrid measurements had an improvement of up to 53% to the fastest pure MPI version. With FS3D, we don't see any improvement for the fastest hybrid runs. However, as FS3D is numerical stable only up to 16 MPI processes, one can gain a 50% speedup. Unfortunately for CDPLit, no improvement for the hybrid version could be observed.

For each of the codes, we observed that the pure OpenMP scalability with 2 threads was in the range of 34% to 79% in comparison to the 1 MPI process versions. This continued to scale up to 147% with 8 threads for XNS and 70% for FS3D. CDPLit, however, failed to scale and lost performance resulting at 17% with 8 threads. However the scalability of pure OpenMP for all codes did not reach any of the MPI parallelizations. This can be explained by the lower parallelization level of the OpenMP implementation and the far more sophisticated MPI implementation.

Additionally, we measured the serial runtime of the MPI-only binary of XNS and FS3D was about the same with the hybrid binaries. However, the CDPLit hybrid binary was more than 10% slower than the MPI only binary. In our opinion, this explains why the hybrid CDPLit did not improve the MPI-only version, as the speedup from OpenMP is countered by the slower initial runtime.

Again, runtime variations were in the range of 5% to 25% for all codes and test sets.

7 Conclusion

From our measurements, we can conclude that for XNS, FS3D and CDPLit, it is never beneficial to use all 8 cores of a node for either threads or MPI processes. We also showed that adding OpenMP to existing codes can improve scalability and efficiency on clusters with shared memory nodes. However, there is no guarantee. Additionally, we observed that if OpenMP scales with 1 MPI process, it continues to scale with larger MPI counts, as long as there is sufficient data within a MPI process. In addition to the MPI and OpenMP scalability, OS jitter has a noticeable impact with variation of runtime up to 24%.

For future work, one can improve the CDPLit OpenMP implementation, as we recently found out that the Intel Compiler implements workshare constructs for Fortran array operations as a single construct using only one thread for the array operation. Furthermore, the reasons for the end of scalability for XNS, FS3D and CDPLit are unknown, and in depth performance analyses are necessary to spot the causes.

References

1. OpenMP Architecture Review Board: OpenMP application program interface (May 2008), <http://www.openmp.org/mp-documents/spec30.pdf>
2. Behr, M., Arora, D., Benedict, N.A., O'Neill, J.J.: Intel compilers on linux clusters. Intel Developer Services online publication (October 2002)

3. Rieber, M.: Numerische modellierung der dynamik freier grenzflächen in zweiphasenströmungen. In: Fortschritt-Berichte VDI: Reihe 7, Strömungstechnik 459 (2004)
4. Zeng, P., Sarholz, S., Iwinsky, C., Binninger, B., Peters, N., Herrmann, M.: Simulation of primary breakup for diesel spray with phase transition. In: Ropo, M., Westerholm, J., Dongarra, J. (eds.) EuroPVM/MPI 2009. LNCS, vol. 5759, pp. 313–320. Springer, Heidelberg (2009)
5. Ham, F., Mattsson, K., Iccarino, G.: Accurate and stable finite volume operators for unstructured flow solvers (2006)
6. Lin, Y., Terboven, C., an Mey, D., Copty, N.: Automatic Scoping of Variables in Parallel Regions of an OpenMP Program. In: Chapman, B.M. (ed.) WOMPAT 2004. LNCS, vol. 3349, pp. 83–97. Springer, Heidelberg (2005)
7. Rabenseifner, R.: Hybrid parallel programming HPC plattforms. In: Fifth European Workshop on OpenMP, Aachen, Germany (2003)

A Maxwell-Schrödinger-Plasma Model and Computing Aspects for Intense, High Frequency and Ultrashort Laser-Gas Interaction

Emmanuel Lorin¹ and André D. Bandrauk²

¹ School of Mathematics and Statistics, Carleton University, Ottawa, Canada
`elorin@math.carleton.ca`

² Laboratoire de chimie théorique, Faculté des Sciences,
Université de Sherbrooke, Québec, J1K 2R1, Canada

Abstract. This paper is devoted to the improvement of a numerical Maxwell-Schrödinger system, modeling the intense, high frequency and ultrashort laser-gas interaction and propagation, and previously presented in [3]. To this original model is added an equation of evolution of the free electron density, allowing then to describe precisely the current density in the Maxwell equations. Free electrons are obtained by absorption of the wavefunctions at the boundary of their corresponding TDSE computational domain. As a consequence higher intensity and longer pulses can then be considered using this new model compared to [3]. The new computing strategy is then presented, following [4].

1 Introduction

Interaction of ultrashort intense laser pulses with atoms [1] and molecules [2] is the beginning of a new science requiring highly nonlinear nonperturbative theoretical and numerical methods based on the laser-matter interaction through the quantum Time Dependent Schrödinger Equation, TDSE, and the concomitant pulse propagation through Maxwell's equations [3], [4], [5], [6].

This paper follows last year's HPCS2008 paper [4], where a Maxwell-Schrödinger system was presented, modeling the propagation and interaction of intense laser pulses in dense gaseous media. The principle that will be shortly recalled in this paper is to couple many TDSEs (describing the gas action on the laser field) with macroscopic Maxwell's equations (modeling the laser field propagation). This model has been successful in reproducing micro-macro phenomena [3]. However, when the initial pulse duration or the initial pulse intensity increase the Maxwell-Schrödinger model reaches its limit. Indeed for long pulses in particular, the density of free electrons increases leading to the creation of a current. As long as these free electrons are not “too far” from their parent nucleus, the TDSE take them into account. We recall that the Maxwell-Schrödinger equations takes in principle in consideration high order nonlinearities and harmonics. Plasma effects important for describing filamentation for instance [7], [6] are then also included as well, if the pulse duration ($\leq 20\text{fs}$) and/or intensity are small enough ($I \leq$

$10^{15} \text{W}\cdot\text{cm}^{-2}$). As TDSE computational domain size is limited by computing constraints, far free electrons can not be totally included in the TDSE. We then have to model a transition between bound and free electrons, and then to model free electron motions independently of the TDSE's. The modeling of free electron motions is crucial as it generates a current taking part of important phenomena such as filamentation. We then propose in this paper an improvement of the Maxwell-Schrödinger model and their general numerical solutions for taking free electron motions into account. Consequently the Maxwell equations have also to be modified. We then present the important computing aspects for solving this new model.

2 Maxwell-Schrödinger Model

We shortly recall the principle of the model. We work under the dipole approximation assuming that the electric field is constant in space, at the molecule scale. This is valid when the smallest internal wavelengths λ_{\min} of the electromagnetic field are much larger than the molecule size ℓ , that is $\ell = o(\lambda_{\min})$. We denote by $\Omega \subset \mathbb{R}^3$ the bounded space domain with a boundary Γ and $\mathbf{r} = (x, y, z)^T$ the space variable in Ω . At the molecule scale, we will denote by $(\mathbf{r}', R') = (x', y', z', R')^T \in \mathbb{R}^3 \times \mathbb{R}_+$ the space variable (for electrons and ions). The molecular density n , is supposed to be constant in time. The equations we consider are the following:

$$\left\{ \begin{array}{lcl} \partial_t \mathbf{B}(\mathbf{r}, t) & = & -c \nabla \times \mathbf{E}(\mathbf{r}, t) \\ \partial_t \mathbf{E}(\mathbf{r}, t) & = & c \nabla \times \mathbf{B}(\mathbf{r}, t) - 4\pi \partial_t \mathbf{P}(\mathbf{r}, t) \\ \nabla \cdot \mathbf{B}(\mathbf{r}, t) & = & 0 \\ \nabla \cdot (\mathbf{E}(\mathbf{r}, t) + 4\pi \mathbf{P}(\mathbf{r}, t)) & = & 0 \\ \mathbf{P}(\mathbf{r}, t) & = & n(\mathbf{r}) \sum_{i=1}^{\ell} \mathbf{P}_i(\mathbf{r}, t) \\ & = & n(\mathbf{r}) \sum_{i=1}^{\ell} \chi_{\Omega_i}(\mathbf{r}) \int_{\mathbb{R}^3 \times \mathbb{R}_+} |\psi_i(R', \mathbf{r}', t)|^2 \mathbf{r}' d\mathbf{r}' dR' \quad (1) \\ i \partial_t \psi_i(R', \mathbf{r}', t) & = & -\frac{\Delta_{\mathbf{r}'}}{2} \psi_i(R', \mathbf{r}', t) - \frac{\Delta_{R'}}{m_p} \psi_i(R', \mathbf{r}', t) + \\ & & \theta(R', \mathbf{r}') \cdot \mathbf{E}_{\mathbf{r}_i} \psi_i(R', \mathbf{r}', t) + \\ & & \left(V_i(R') + V_c(R', \mathbf{r}') \right) \psi_i(R', \mathbf{r}', t), \quad \forall i \in \{1, \dots, m\} \end{array} \right.$$

In (1), V_c denotes the Coulomb potential, V_i the nuclei potential and θ is a regular vectorial function with compact support \mathcal{D}_1 equal to \mathbf{r}' on a compact set $\mathcal{D}_2 \subset \mathcal{D}_1$.

In (1), Ω_i is the spatial domain associated to ψ_i , wavefunction of the i^{th} TDSE and \mathbf{P}_i denotes the polarization associated to this domain. The space Ω_i contains $n(\mathbf{r}) \Omega_i$ molecules represented by the wavefunction ψ_i of one molecule located in Ω_i . We assume that the spatial support of ψ_i is strictly included in a domain $\omega_i \subset \mathbb{R}^3 \times \mathbb{R}_+$.

Functions χ_{Ω_i} are defined by $\chi \otimes \mathbf{1}_{\Omega_i}$ where $\chi \in \mathcal{C}_0^\infty(\mathbb{R}^3)$ is a plateau function and $\mathbf{1}_{\Omega_i}$ is the characteristic function of Ω_i . Naturally we have $\cup_{i=1}^\ell \Omega_i = \Omega$. The model is limited by the fact that for long pulses for instance, an electron wavefunction can reach the boundary of the computational TDSE box. In that situation it is important to describe precisely what happens in that region. In fact the fourth equation in (II) includes charge conservation, both bound (quantum) and free (ionized) electrons during propagation.

3 Maxwell-Schrödinger-Plasma Model

We start this section by explaining the conditions that are imposed at the boundary of each TDSE computational domain. As presented in [5] several types of boundary conditions can be imposed describing different kinds of physical processes. Also, the approximation of these continuous boundary conditions can lead to many computing difficulties (see [8] for a complete state of the art of boundary conditions and their approximation for the TDSEs). In the following we will impose an absorber at the boundary of the computational domain. The principle is to absorb the part of the wavefunction that reaches the boundary due to laser ionization. The goal is first from the numerical point of view to avoid a spurious numerical reflection of the wave at the boundary. Indeed, a reflecting effect is well-known to occur in that situation (see [3] and again more generally in [8]). When the pulse is not too intense or short enough and when at the same time the TDSE computational box has been chosen large enough, the wavefunction support remains inside the computational domain ($\|\psi(\cdot, t)\|_{L^2} = 1$ for all t): this means that all the electrons are taken into account by the TDSE and then by the polarization \mathbf{P} . In the opposite case, due to memory and CPU time constraints, we absorb at the boundary a part of the wavefunction (the L^2 norm is not conserved anymore in time). The loss $1 - \|\psi(\cdot, t)\|_{L^2}$ corresponds to free electrons, as they are no more attached anymore to a nucleus; their motion is classically driven by the electric field. In that situation it has to be noticed that the polarization in (II) only includes the bound electrons. We then impose absorbing boundary conditions on $\partial\omega_i$ for all $i \in \{1, \dots, m\}$. That is, for ε small enough, we define a regular decreasing function f_ε , from 1 for \mathbf{r}' such that $d(\mathbf{r}', \partial\omega_i) = \varepsilon$, to 0 for all \mathbf{r}' such that $d(\mathbf{r}', \partial\omega_i) = 0$ (that is $\mathbf{r}' \in \partial\omega_i$). We then multiply ψ_i by f_ε for all \mathbf{r}' such that $d(\mathbf{r}', \omega_i) \leq \varepsilon$. Because of the absorption at the boundary, $\|\psi_i(t)\|_{L^2(\omega_i)}$ is no anymore invariant in time, and then $\partial_t \|\psi_i(t)\|_{L^2(\omega_i)}$ is negative or zero, see Fig. II. We then add to the original system the current density equation involving the evolution of the free electron density:

$$\partial_t \rho(\mathbf{r}, t) + \nabla \cdot (\rho \mathbf{v}_E)(\mathbf{r}, t) = -n(\mathbf{r}) \sum_{i=1}^\ell \chi_{\omega_i}(\mathbf{r}) \partial_t \|\psi_i(t)\|_{L^2(\omega_i)}. \quad (2)$$

The source term represents the free electron (unbound) “production”. Note in particular that when molecules are totally ionized in ω_i , the rhs of (2) is zero.

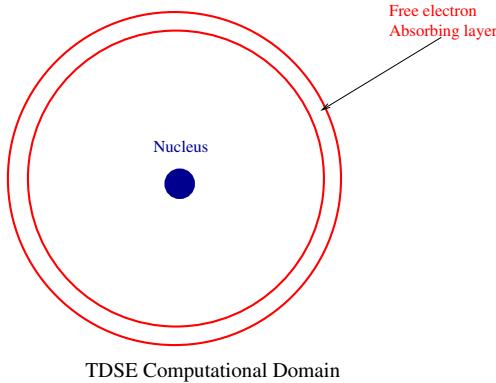


Fig. 1. Free electron absorption for domain $\partial\omega_i$ and wavefunction ψ_i

Inversely if the electric field intensity is too low to produce ionization, the time derivative of $\|\psi_i(t)\|_{L^2(\omega_i)}$ is zero corresponding to no free electron production in Ω_i . The time process is more generally the following. When locally in ω_i the electric field is intense enough then $-n(\mathbf{r})\partial_t\|\psi_i(t)\|_{L^2(\omega_i)}$ free electrons are released in Ω_i . Note that this free electron motion model is very similar to classical ones such as in [6] or [7] where however the free electron production is given by a macroscopic term involving the number of photons in the multi-ionization. Free electrons are driven by \mathbf{E} at a velocity \mathbf{v}_E . However, in practice, for high frequency fields (respectively for fields initially polarized transversely to the propagation direction z), we can neglect the motion in x and y (respectively in z) so that $\mathbf{v}_E \sim \mathbf{0}$. The current density equation then is (neglecting the ponderomotive forces)

$$\partial_t \mathbf{J} + \frac{1}{\tau_c} \mathbf{J} = \frac{e^2}{m_e} \rho \mathbf{E} \quad (3)$$

where $e = m_e = 1$ in *a.u.* and τ_c is the collision time. Moreover, when very short pulses are considered (less than $20fs$) the collision term can also be neglected in (3). Considering the free electrons are released by ionization the Gauss equation then becomes

$$\nabla \cdot (\mathbf{E}(\mathbf{r}, t) + 4\pi \mathbf{P}(\mathbf{r}, t)) = \rho(\mathbf{r}, t).$$

In Maxwell's equations, the time evolution of the electric field becomes

$$\partial_t \mathbf{E}(\mathbf{r}, t) = c \nabla \times \mathbf{B}(\mathbf{r}, t) - 4\pi (\partial_t \mathbf{P}(\mathbf{r}, t) + \mathbf{J}(\mathbf{r}, t))$$

where this time \mathbf{P} denotes the polarization associated to bound electrons only. Note that at this point, this model does not allow free electrons at the boundary to recombine with nuclei. However electrons in each domain will recollide and recombine through the equations with parent ions thus generating high order harmonics [2] from which one obtains "attosecond" pulses [9].

4 Computing Aspects

Following the strategy presented in [4] for solving the Maxwell-Schrödinger equations, the extension to Maxwell-Schrödinger-Plasma equations is as follows. As in [4], we suppose that there is one processor per node (in practice, 2 processors per node on `mammouth`, <http://ccs.usherbrooke.ca>) and that we use N_p processors. From time t^n to t^{n+1} here is the following computational scheme.

1. From time t^n to t^{n^*} , we solve ℓ TDSE's on m processors. Each TDSE is solved sequentially (ℓ/m TDSE's per processor). We then deduce the updated polarization \mathbf{P} , as well as the free electron density ρ , from the absorption of the wavefunctions at the TDSE computational domains.
2. Local polarization and free electron density are sent to the nodes in charge of the Maxwell equation computation, more precisely, the nodes in charge of the gas regions.
3. From time t^{n^*} to $t^{n^{**}}$, the current density \mathbf{J} , is updated by solving (B). An order two scheme is required in order to conserve the total 2 order of the overall method. Although the current density is only updated by the processors in charge of the gas regions (leading to a certain waste of time, as the other processors do not work during this step), due to the simplicity of the equations, it is a very fast step that does not slow down the overall procedure.
4. From time $t^{n^{**}}$ to t^{n+1} , we finally solve by domain decomposition the Maxwell equations as in [4], from time $t^{n+1/2}$ to t^n on $N_p - m$ processors.
5. Data storage is comparable to Maxwell-Schrödinger (see [4]) as free electron density is a scalar number.

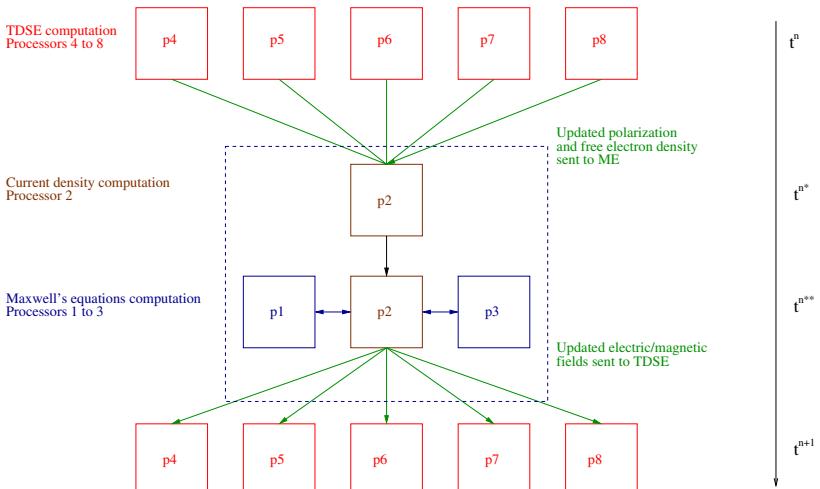


Fig. 2. Parallelism Principle

The scheme can be summarized as follows Fig. 2 where $N_p = 8$, $m = 3$ and the gas is treated at the macroscopic level (Maxwell) by one processor. The efficiency of the overall code remains comparable to [4]. Indeed and as said above the current density is computed extremely fast due to the simplicity of the differential equation.

5 Conclusion

We have proposed in this paper, an improvement of the Maxwell-Schrödinger system for modeling intense, high frequency and short laser-gas interaction and propagation. An equation on current density, deduced from free electron motion is added to the model allowing to improve the effect of the plasma of free electrons created by molecular ionization. This approach, both quantum for bound electrons and classical for ionized free electrons, will allow us to consider more intense and longer pulses for which free electrons have a crucial effect: defocusing of the beam, potentially counter-balanced by the nonlinear self-focusing effects (such as Kerr) leading to the creation of filamentation [7], [6]. Such filamentation is of interest to attosecond pulse generation due to the compression of incident pulses by highly nonlinear response of matter at high laser intensities.

References

1. Brabec, T., Krausz, F.: Intense Few Cycle Light Pulses. *Rev. Mod. Phys.* 72, 545–591 (2000)
2. Bandrauk, A.D., Barmaki, S., Kamta, G.L.: Molecular High Order Harmonic Generation. In: Yamanouchi, K. (ed.) *Progress in Ultrafast Intense Laser Science*, ch. 9, vol. III. Springer, NY (2008)
3. Lorin, E., Chelikowski, S., Bandrauk, A.: Numerical Maxwell-Schrödinger model for laser-molecule interaction and propagation. *Comput. Phys. Comm.* 177(12), 908–932 (2007)
4. Lorin, E., Bandrauk, A.: Numerical Maxwell-Schrödinger model for laser-molecule interaction and propagation. In: IEEE Proc. HPCS 2008, pp. 4–8 (2008)
5. Lorin, E., Chelikowski, S., Bandrauk, A.: Numerical Maxwell-Schrödinger model for laser-molecule interaction and propagation. *Num. Meth. for Partial Differential Equations* 25(1), 110–136 (2009)
6. Couairon, A., Mysyrowicz, A.: Organizing multiple femtosecond filaments in air. *Phys. Report.* 41(3), 47–189 (2007)
7. Bergé, L., Skupin, S., Nuter, R., Kasparian, J., Wolf, J.-P.: Ultrashort filaments of light in weakly ionized, optically transparent media. *Reports on Progress in Physics* 70(10), 1633–1713 (2007)
8. Antoine, X., Arnold, A., Besse, C., Ehrhardt, M., Schädle, A.: A review of transparent and artificial boundary conditions techniques for linear and nonlinear Schrödinger equations. *Comm. Comput. Phys.* 4(4), 729–796 (2008)
9. Corkum, P., Krausz, F.: Attosecond Science. *Nature Phys.* 3, 381–387 (2007)

The Implementation of Polarizable and Flexible Models in Molecular Dynamics Simulations

Shihao Wang and Natalie M. Cann

Department of Chemistry, Queen's University, Kingston, Ontario, Canada K7L 3N6
ncann@chem.queensu.ca

Abstract. We discuss a new methodology for implementing polarizable and flexible molecular models - the fluctuating charge and intramolecular potential (fCINTRA) method - in Molecular Dynamics (MD) simulations. An example has been provided for ethanol. In these models, all potential parameters depend on the local electrostatic field generated by the other molecules in the system. A methodology for extracting field-dependent intramolecular potentials from *ab initio* calculations is discussed, and the parameters controlling the energetics of intramolecular motion are directly coupled to the field experienced by the atoms in a molecule. Variability in the atomic charges is introduced via the fluctuating charge model of Rick *et al.* [S. Rick, *et al.*, J. Chem. Phys. 1994, 101, (7), 6141-6156.]. Atomic charge fluctuations are much faster than atomic motion and, for practical reasons, a multiple time steps algorithm is required. In the implementation of MD simulations for this model, the Message Passing Interface (MPI) has been used. The simulation algorithm is complex for the fCINTRA model. However, with the help of load sharing, minimization of interprocessor communications, and code optimization, the overall simulation time is acceptable.

Keywords: Molecular Dynamics Simulation, polarizable and flexible model, Message Passing Interface.

1 Introduction

In molecular dynamics (MD) simulations, non-polarizable models have been widely used [1,2]. In these models, the potential parameters are defined at the outset of the simulation and do not change throughout. Non-polarizable models are practical to implement and efficient to use but they have obvious limitations in that they are unable to accurately treat molecules in significantly differing environments [1]. For example, Shirts *et al.* [3] found that these force fields tend to underestimate the solubility of amino acid side chain analogs. It is clear that, when a molecule is placed in different environments, its properties, such as the atomic charge distribution or energetic costs for a conformational change, can be very different. Non-polarizable models neglect these differences and set these properties to be constant regardless of the molecular environment. Quantum dynamics methods [4] are the ideal approach for including molecular response since the electrons are explicitly present. However, these methods

are currently impractical for many systems of chemical interest. To be precise, they are restricted to simulations of a limited number of particles over a relatively short time frame [5]. Polarizable *classical* models, where the electronic response is parameterized into the model, can treat the influence of environment on molecules more accurately than non-polarizable models while remaining practical. However, because the computational costs are higher than for non-polarizable models, polarizable models are not widely used.

There have been continuing efforts to develop polarizable models and several methods have been proposed [6,11]. Gao *et al.*⁶ developed the polarizable intermolecular potential function (PIPF) for liquid alcohols, where the total energy of the system consists of a pairwise term and a nonadditive polarization term. Noskov *et al.* [7,9] developed a polarizable model based on Drude oscillators. In their model, the partial charge on a heavy atom, such as oxygen, is redistributed among a set of massless Drude oscillators connected by a harmonic spring. Svirshchev *et al.* [11] developed a polarizable point-charge (PPC) model for water. This model introduced electrostatic field-dependence to atomic charges and intramolecular structures. Rick *et al.* [10], introduced a fluctuating charge (FC) model that uses an extended Hamiltonian approach to allow for instantaneous changes in atomic charges.

Although these models allow for fluctuations in atomic charges, they all neglect field dependence in the remaining potential parameters. To our knowledge, a field-dependent intramolecular potential has been developed only once previously, for water [12]. In that model, the HH and OH stretching potentials of water depended on the magnitude of the electrostatic field on oxygen. This model is specific to water and all other potential parameters, including the atomic charges, are field-independent. In our previous paper, we discussed the development of a new polarizable and flexible model, the fluctuating charge and intramolecular potential (fCINTRA) model [13]. This model includes field dependence in both the atomic charges and the intramolecular degrees of freedom. In this article, full details on the implementation of the fCINTRA model are provided. Our previous work [13] focused specifically on ethanol but in this article we generalize to other molecules and to molecules with internal constraints (rigid bonds, for example). Field dependent symmetry considerations and implementation details are also provided. Section 2 briefly describes the fCINTRA model. The implementation of the model is presented in Section 3.

2 Models

A fully field-dependent potential is written as

$$U(\vec{E}) = U^{el}(\vec{E}) + U^{intra}(\vec{E}) + U^{LJ}(\vec{E}), \quad (1)$$

where \vec{E} is the electrostatic field, which is defined as

$$\vec{E}(\vec{r}_{i\alpha}) = \sum_{\beta=1}^M \sum_{j=1}^{N_\beta} \sum_{\vec{n}} \frac{Q_{j\beta}}{|\vec{r}_{i\alpha j\beta} + \vec{n}L|^2} \vec{r}_{i\alpha j\beta} \quad (2)$$

where $\vec{n} = (n_x, n_y, n_z)$ with n_x, n_y, n_z integers, L is the length of the simulation cell, and j is an atom in molecule β . $U^{\text{LJ}}(\vec{E})$ is the Lennard-Jones (LJ) energy,

$$U^{\text{LJ}}(\vec{E}) = \sum_{i\alpha} \sum_{j\beta} 4\epsilon_{ij}(\vec{E}) \left[\left(\frac{\sigma_{ij}(\vec{E})}{r_{i\alpha j\beta}} \right)^{12} - \left(\frac{\sigma_{ij}(\vec{E})}{r_{i\alpha j\beta}} \right)^6 \right], \quad (3)$$

where $\epsilon_{ij}(\vec{E})$ and $\sigma_{ij}(\vec{E})$ are the field-dependent LJ parameters. A cutoff radius, which is slightly less than half of the simulation box length ($r_{\text{cut}}=L/2.05$), is applied and the LJ potential is shifted at truncation [2]. $U^{\text{el}}(\vec{E})$ is the electrostatic energy, discussed below, and $U^{\text{intra}}(\vec{E})$ is the intramolecular potential. The intramolecular potential typically consists of four parts:

$$U^{\text{intra}}(\vec{E}) = U^{\text{st}}(\vec{E}) + U^{\text{be}}(\vec{E}) + U^{\text{tor}}(\vec{E}) + U^{\text{imp}}(\vec{E}). \quad (4)$$

although some parametrizations [14,15] include more terms. The generalization to these potentials is straightforward and will not be considered here. Stretching and bending are typically described by harmonic potentials as

$$U^{\text{st}}(\vec{E}) = \sum_{is} k_{s;is}(\vec{E})(r_{is} - r_{e;is}(\vec{E}))^2, \quad (5)$$

$$U^{\text{be}}(\vec{E}) = \sum_{ib} k_{\theta;ib}(\vec{E})(\theta_{ib} - \theta_{e;ib}(\vec{E}))^2, \quad (6)$$

where $k_{s;is}(\vec{E})$ and $k_{\theta;ib}(\vec{E})$ are the field-dependent stretching and bending force constants, respectively, and $r_{e;is}(\vec{E})$ and $\theta_{e;ib}(\vec{E})$ are the equilibrium bond length and angle, respectively. Bending and bond stretching are energetically costly, relative to thermal energy (i.e. RT) and deviations from the equilibrium values are generally small. For this reason, a harmonic potential is satisfactory for most molecules, but this is not the case for torsions where the full rotation over 360 degrees should be considered. The field-dependent torsional potential is described by an extended Ryckaert-Bellemand potential [16]

$$U^{\text{tor}}(\vec{E}) = \sum_{it} \left(\sum_{i=0}^6 C_{it;i}(\vec{E}) \cos^i (\varphi_{it} + \varphi_{0,it}(\vec{E})) \right), \quad (7)$$

where φ_{it} is the dihedral angle and $\varphi_{0,it}(\vec{E})$ is the corresponding phase shift. The latter is important in representing the field response of the molecule. Specifically, the potential in the presence of a field may have a lowest energy conformer at a different

torsional angle than the zero-field case and the potential must allow for this shift. Improper torsion potentials have the form

$$U^{imp}(\vec{E}) = \sum_{im} k_{i,im}(\vec{E})(t_{im} - t_{e,im}(\vec{E}))^2 , \quad (8)$$

where t_{im} , $t_{e,im}(\vec{E})$, and $k_{i,im}(\vec{E})$ are the improper torsion angle, the equilibrium value of the angle, and the force constant, respectively.

For ethanol, intramolecular motion was represented by 8 stretch, 13 bend, and 2 torsion potentials [13]. The model was fully flexible but the impact of constraints, such as fixed bond lengths, are discussed below. After extensive B3LYP/6-311++G** calculations for ethanol in the presence of a variety of fields, we found that the stretching force constants, bending force constants, and bond lengths were only weakly field dependent, and therefore we regarded them as field-independent. The equilibrium bond angles and torsional potential coefficients were more strongly field sensitive and their field dependence was included in the model. The torsional potential for ethanol was defined as

$$U^{tor}(\vec{E}) = \sum_{it=1}^2 \left(\sum_{i=0}^6 C_{it,i}(\vec{E}) \cos^i \varphi_{it} + C_{it,7}(\vec{E}) \cos^3(\varphi_{it} - 90^\circ) + C_{it,8}(\vec{E}) \cos^5(\varphi_{it} - 90^\circ) \right) , \quad (9)$$

a slight variation from Eq. [7].

There are many ways to include field-dependence in the potential parameters. For instance, the parameters can be given a field dependence based on empirical arguments, they can depend only on the field at the center of mass, they can be expanded in linear combinations of electrostatic fields on each atom, and so on. In order to be used in MD simulations, the terms should be easy to evaluate and independent of the coordinate system, so that the local to global coordinate transformation is not required. We recommend “structural” analogs to expand the coefficients. For example,

$$| E_X^1 | , \quad (10)$$

$$| E_{XY}^{12} | = | \vec{E}_X^1 \| \vec{E}_Y^2 | , \quad (11)$$

$$E_{XY}^{12} = \vec{E}_X^1 \bullet \vec{E}_Y^2 , \quad (12)$$

$$E_{XYZ}^{123} = (\vec{E}_X^3 - \vec{E}_Y^2) \bullet (\vec{E}_Y^2 - \vec{E}_Z^1) , \quad (13)$$

$$\hat{E}_{XYZ}^{123} = (\vec{E}_Z^3 \times \vec{E}_Y^2) \bullet (\vec{E}_Y^2 \times \vec{E}_X^1) , \quad (14)$$

$$E_{XYZA}^{1234} = ((\bar{E}_A^4 - \bar{E}_Z^3) \times (\bar{E}_Z^3 - \bar{E}_Y^2)) \bullet ((\bar{E}_Z^3 - \bar{E}_Y^2) \times (\bar{E}_Y^2 - \bar{E}_X^1)) . \quad (15)$$

where X, Y, Z, and A are atoms in the molecule. These terms are field equivalents of bond angles, torsions, and so on. Since they involve only field magnitudes or reduce to dot and cross products, they don't depend on the chosen coordinate system. For large molecules, where there are many atoms and the number of coefficients of the form shown in Eqs. (10)-(15) is high, a subset may be chosen based on statistical analysis of the fitted potentials.

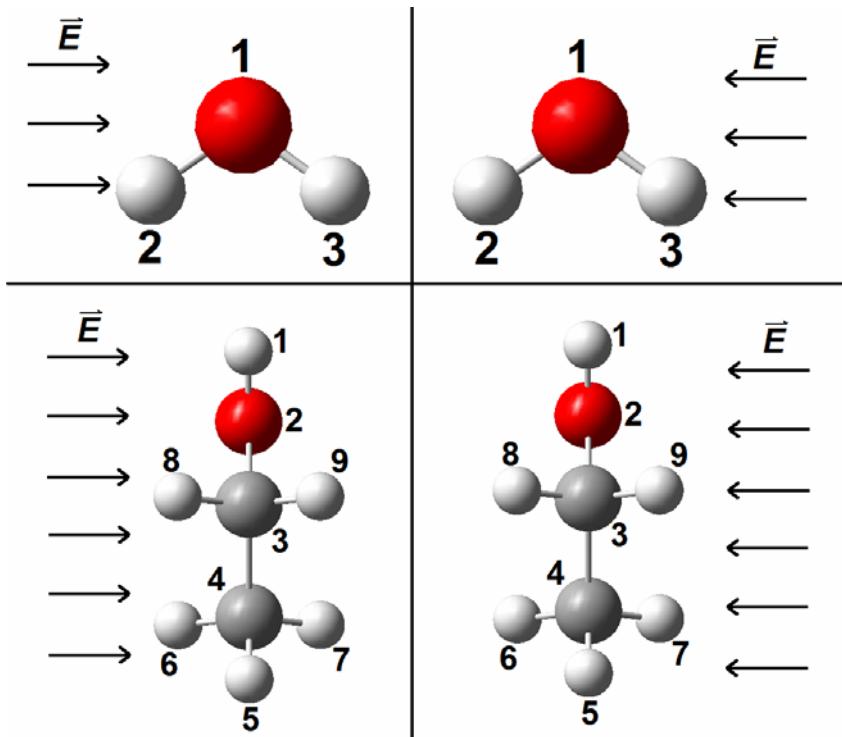


Fig. 1. Molecules under symmetric electrostatic fields. The upper and lower panels show water and ethanol molecules, respectively, under symmetric fields.

The symmetry of the molecules should be considered in the selection of terms in Eqs. (10)-(15). Specifically, if the molecule possesses symmetry elements then the symmetry impacts the field response. For example, if two electrostatic fields are symmetric about a symmetry plane in a molecule, their influence on the molecule should be the same, as demonstrated in Fig. 1. In the lower panels of the figure, the ethanol molecule is in two symmetric fields as shown on the left and the right. Thus, atom 8 in the left panel and atom 9 in the right panel should have the same charge, and bond angles H(8)-C(3)-O(2) in the left panel and H(9)-C(3)-O(2) in the right panel

should have the same equilibrium value. This symmetry has been taken into account for ethanol [13]. For instance, since H(8) and H(9) are related by symmetry, we defined

$$E_{HOCH}^{123X} = E_{HOCH}^{1238} + E_{HOCH}^{1239} . \quad (16)$$

and, by design, the field response is symmetric for H(8) and H(9). In general, the symmetry elements possessed by the molecule in its equilibrium structure must be reflected in the field-dependent terms of the expansion. In choosing a particular expansion according to the terms in Eqs. (10)-(15), molecular symmetry must be considered and enforced.

To develop an effective polarizable and flexible model, some estimate of the variation and frequency of fields experienced by molecules in the bulk is required. For ethanol, molecular dynamics simulations with a non-polarizable model were first used to estimate “typical” electrostatic fields. These collected fields were analyzed and a representative subset selected to perform *ab initio* calculations. The *ab initio* calculations must reproduce the selected fields without the inclusion of many neighboring molecules. This can be accomplished by placing point charges around a single molecule, with the magnitude and position of these charges chosen to correctly reproduce the fields at each atom. However, it is important for the charges to be far from the molecule so that they do not accumulate electron density.

In the absence of geometric constraints in the *ab initio* calculations, the molecule will rotate and translate in the field but this will not happen within the MD simulation where neighboring molecules provide a positional constraint and the field varies instantaneously. For ethanol, the *ab initio* calculations included constraints on the carbon, oxygen and hydrogen (C(3), O(2) and H(1)) positions. Field dependent intramolecular potentials have not been derived for larger molecules but the obvious approach – a fully constrained geometry – will not provide information on energetic costs for intramolecular motion.

In simulations, with hundreds of molecules and millions of time steps, atoms will sometimes experience fields that are outside of the range of the parameterization of the field-dependent coefficients. Although this will be infrequent if the original fields were diverse and representative of the molecular environment, and the model may still perform well regardless, these “out of range” fields may sometimes result in unphysical potentials. In order to prevent this, a damping function for each field-dependent coefficient was implemented for ethanol [13] to constrain the parameters to a specified interval. Damping will likely be inevitable for all fCINTRA models unless the field dependence of the potential parameters in Eqs. (5)-(8) is such that all possible fields are naturally included by the parameterization. It is not clear, at present, how one could accomplish this.

The atomic charge distribution can also be affected by the molecular environment. Many ways to include charge variability have been proposed [7,10,11]. We adopt the FC model [10] to treat the charge fluctuations since this model is efficient, readily applicable to large molecules, and based on intuitive physical arguments. This model uses an extended Hamiltonian approach to allow the molecules to respond to their

environment and doesn't require any iterations to achieve self-consistency – a draw-back to several other approaches [7,11] to fluctuating charges.

In the FC model, the electrostatic energy of a system of M molecules with N atoms per molecule is defined as [10]:

$$U^{el} = \sum_{\alpha=1}^M \sum_{i=1}^{N_\alpha} \left(E_i(0) + \tilde{\chi}_i^0 Q_{i\alpha} + \frac{1}{2} J_{ii}^0 Q_{i\alpha}^2 \right) + \sum_{i\alpha < j\beta} J_{ij}(r_{i\alpha j\beta}) Q_{i\alpha} Q_{j\beta} . \quad (17)$$

where i is an atom in molecule α and j belongs to molecule β . The first term in Eq. (17) is the energy for charging each atom in the molecule. $E_i(0)$ and $\tilde{\chi}_i^0$ are the ground state energy and the electronegativity per unit charge of atom i , respectively and J_{ii}^0 is twice the hardness of the electronegativity of the atom. The first term is effectively a Taylor expansion truncated to second order, and might be further expanded for larger molecules [17]. The second term in Eq. (17) is the electrostatic energy between all pairs of atoms in the system and $J_{ij}(r_{i\alpha j\beta})$ is the Coulomb interaction.

In general, $J_{ij}(r_{i\alpha j\beta})$ is simply equal to $\frac{1}{r_{i\alpha j\beta}}$ and the second term in Eq. (17) is

a typical, point charge Coulomb repulsion term. For atoms that are in the same molecule and are separated by fewer than four bonds, there are two common ways to evaluate $J_{ij}(r_{i\alpha j\beta})$. One is to use a Coulomb overlap integral [17] between Slater orbitals and an alternative approach [18] is to use a much simpler approximation that relates $J_{ij}(r_{i\alpha j\beta})$ to J_{ii}^0 . We found that the latter approximation incorrectly predicted the fluid structure for bulk ethanol and underestimated the average dipole moment [13]. These results suggest that the use of the more accurate form [17] is necessary.

The parameters $\tilde{\chi}_i^0$ and J_{ii}^0 may be obtained from experiment, from empirical relationships, or from *ab initio* calculations. For the latter, the molecule is placed in a number of different fields and the atomic charges are collected. The FC parameters are then extracted by fitting atomic charges to Eq. (17). Attempts have been made [18] to evaluate transferable values for these parameters in order to generalize and broaden the applicability of the FC method. However, our study on ethanol showed that the transferable FC parameters predicted unphysical atomic charges on some atoms. These parameters, although they are attributed to the individual atoms, reflect the atom *within the molecule*. The transferability of the parameters relies on some insensitivity to molecular environment and our results suggest that an improved accounting of molecular context is required. For instance, carbonyl, ether, and alcohol oxygens should have different parameters. Further developments of transferable FC parameters are certainly warranted and would encourage widespread use of polarizable models.

The model development discussed above relies heavily on *ab initio* calculations. Specifically, the FC parameters in Eq. (17) and the field-dependent intramolecular potential parameters in Eqs. (5)-(8) are exclusively obtained from *ab initio* calculations on single molecules in the presence of a field. As with non-polarizable potentials, the intermolecular potentials are limited to pairwise terms so that *effective* potentials are required. The easiest way to refine the fCINTRA potential for bulk fluids is via the Lennard-Jones parameters. In principle, these may be taken from literature potentials [18,19] but the accuracy of the molecular representation will improve if these are refined based on comparisons with experiment. Properties such as the self-diffusion coefficient, average molecular dipole moment, dielectric constant, and interatomic distributions extracted from neutron diffraction experiments provide reasonable model assessment criteria, if they are available.

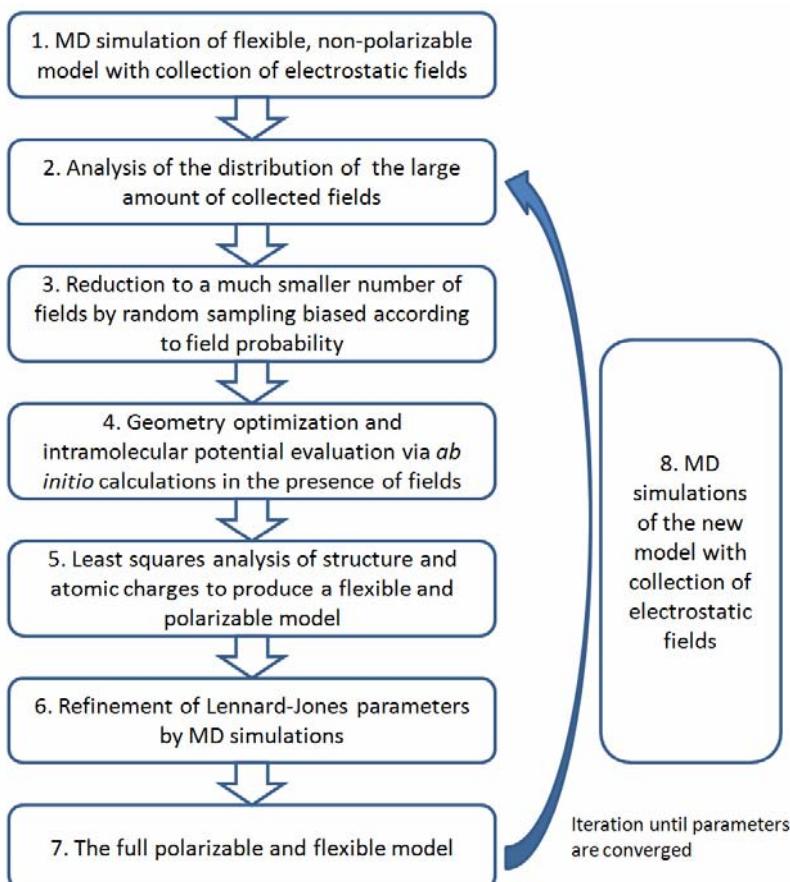


Fig. 2. Flow chart showing the ideal procedure of developing the fCINTRA model. Based on a flow chart published elsewhere [13].

In the development of the model, the field-response can be further refined by reiterating the parameterization process. As shown in Fig.2, the model parameters are developed based on electrostatic fields collected in the original MD simulations of a non-polarizable model. Since the electrostatic fields are sensitive to models, non-polarizable models will not generate the same field distributions as polarizable models. It follows that the model parameters can be further refined from the electrostatic fields collected from the MD simulations with the fCINTRA model. Ideally, this procedure should be iterated until the parameters are fully converged, but this will be a very time consuming process. Fig.2 shows the ideal, iterative procedure for developing polarizable and flexible models.

3 Implementation

3.1 Force Calculations

The forces under which the atoms move are directly evaluated from the gradient of the non-polarizable potential. In the fCINTRA model, the explicit inclusion of electrostatic fields in the potential makes the force calculation more complicated [13]. Positions are not only directly included in the potentials, but also indirectly included through electrostatic fields. The FC model for fluctuating charges also requires a calculation of forces to govern charge fluctuations. Therefore, the force calculations in the fCINTRA model are much more complicated than those for non-polarizable models.

In force calculations for positions, the forces are evaluated from

$$\begin{aligned} F_{\vec{r}}(\vec{r}, Q) &= -\vec{\nabla}_{\vec{r}_{i\alpha}} U(\vec{E}) \\ &= -\frac{\partial U^{el}}{\partial \vec{r}_{i\alpha}} - \frac{\partial U^{LJ}}{\partial \vec{r}_{i\alpha}} - \frac{\partial U^{LJ}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial \vec{r}_{i\alpha}} - \frac{\partial U^{intra}}{\partial \vec{r}_{i\alpha}} - \frac{\partial U^{intra}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial \vec{r}_{i\alpha}}, \end{aligned} \quad (18)$$

where the five terms are electrostatic force, explicit LJ force, implicit LJ force, explicit intramolecular force, and implicit intramolecular force. The forces that govern atomic charge fluctuations are obtained from

$$\begin{aligned} F_Q(\vec{r}, Q) &= -\nabla_Q U(\vec{E}) = -\frac{\partial U}{\partial Q_{i\alpha}} + \frac{1}{N_\alpha} \sum_{j=1}^{N_\alpha} \left(\frac{\partial U}{\partial Q_{j\alpha}} \right) \\ &= -\frac{\partial U^{el}}{\partial Q_{i\alpha}} + \frac{1}{N_\alpha} \sum_{j=1}^{N_\alpha} \left(\frac{\partial U^{el}}{\partial Q_{j\alpha}} + \frac{\partial U^{intra}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial Q_{j\alpha}} \right) - \frac{\partial U^{intra}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial Q_{i\alpha}} \end{aligned} \quad (19)$$

where the first term is the analog of the electrostatic force, the second term (the summation) comes from the charge neutrality constraint imposed on the molecule, and the last term is the implicit dependence of the intramolecular potential on atomic charges.

The terms in Eqs. (18)-(19) are evaluated as follows:

$$\begin{aligned} \frac{\partial U^{el}}{\partial \vec{r}_{i\alpha}} = & \frac{2\pi}{V} Q_{i\alpha} \sum_{\beta=1}^M \sum_{j=1}^{N_\beta} \sum_{\vec{k}} \frac{Q_{j\beta}}{k^2} \exp\left(-\frac{k^2}{4\alpha}\right) \vec{k} \sin(\vec{k} \bullet \vec{r}_{i\alpha j\beta}) \\ & + Q_{i\alpha} \sum_{\beta=1}^M \sum_{j=1}^{N_\beta} (1 - \delta_{ij} \delta_{\alpha\beta}) \sum_{\vec{k}} Q_{j\beta} \frac{\operatorname{erfc}(\alpha r_{i\alpha j\beta}) + \frac{2\alpha r_{i\alpha j\beta}}{\sqrt{\pi}} \exp(-\alpha^2 r_{i\alpha j\beta}^2)}{r_{i\alpha j\beta}^3} \vec{r}_{i\alpha j\beta} \\ & - Q_{i\alpha} \sum_{j=1}^{N_\alpha} (1 - \delta_{ij}) \frac{Q_{j\alpha}}{r_{i\alpha j\alpha}^3} \vec{r}_{i\alpha j\alpha}, \end{aligned} \quad (20)$$

where δ'_{ij} in the last term is zero for atoms that are separated by four or more bonds,

and one otherwise. The electrostatic force on charge is evaluated as

$$\begin{aligned} \frac{\partial U^{el}}{\partial Q_{i\alpha}} = & \frac{2\pi}{V} \sum_{\beta=1}^M \sum_{j=1}^{N_\beta} \sum_{\vec{k}} \frac{Q_{j\beta}}{k^2} \exp\left(-\frac{k^2}{4\alpha}\right) \cos(\vec{k} \bullet \vec{r}_{i\alpha j\beta}) \\ & + \sum_{\beta=1}^M \sum_{j=1}^{N_\beta} (1 - \delta_{ij} \delta_{\alpha\beta}) \sum_{\vec{k}} Q_{j\beta} \frac{\operatorname{erfc}(\alpha r_{i\alpha j\beta})}{r_{i\alpha j\beta}} \\ & + \sum_{j=1}^{N_\alpha} (1 - \delta_{ij}) \frac{Q_{j\alpha}}{r_{i\alpha j\alpha}^3} - 2\sqrt{\frac{\alpha}{\pi}} Q_{i\alpha}. \end{aligned} \quad (21)$$

The explicit LJ force is

$$\frac{\partial U^{LJ}}{\partial \vec{r}_{i\alpha}} = - \sum_{j\beta} \frac{24\epsilon_{ij}(\vec{E})}{r_{i\alpha j\beta}^2} \left[2 \left(\frac{\sigma_{ij}(\vec{E})}{r_{i\alpha j\beta}} \right)^{12} - \left(\frac{\sigma_{ij}(\vec{E})}{r_{i\alpha j\beta}} \right)^6 \right] \vec{r}_{i\alpha j\beta}, \quad (22)$$

and the implicit LJ force is

$$\begin{aligned} \frac{\partial U^{LJ}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial \vec{r}_{i\alpha}} = & \sum_{j\beta} \left(\frac{\partial U^{LJ}}{\partial \epsilon_{j\beta}} \cdot \frac{\partial \epsilon_{j\beta}}{\partial \vec{E}_{j\beta}} + \frac{\partial U^{LJ}}{\partial \sigma_{j\beta}} \cdot \frac{\partial \sigma_{j\beta}}{\partial \vec{E}_{j\beta}} \right) \frac{\partial \vec{E}_{j\beta}}{\partial \vec{r}_{i\alpha}} \\ = & \sum_{j\beta} \left\{ 4 \left[\left(\frac{\sigma_{ij}}{r_{i\alpha j\beta}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{i\alpha j\beta}} \right)^6 \right] \frac{\partial \epsilon_{j\beta}}{\partial \vec{E}_{j\beta}} + \frac{24\epsilon}{\sigma} \left[2 \left(\frac{\sigma_{ij}}{r_{i\alpha j\beta}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{i\alpha j\beta}} \right)^6 \right] \frac{\partial \sigma_{j\beta}}{\partial \vec{E}_{j\beta}} \right\} \frac{\partial \vec{E}_{j\beta}}{\partial \vec{r}_{i\alpha}} \end{aligned} \quad (23)$$

For ethanol the LJ parameters, ε_{ij} and σ_{ij} , were considered as field-independent, and

therefore $\frac{\partial U^{\text{LJ}}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial \vec{r}_{i\alpha}} = 0$. Intramolecular forces are calculated as follows

$$\frac{\partial U^{st}(\vec{E})}{\partial \vec{r}_{i\alpha}} = \sum_{is} 2k_{s;is}(\vec{E})(r_{is} - r_{e;is}(\vec{E})) \frac{\partial r_{is}}{\partial \vec{r}_{i\alpha}}, \quad (24)$$

$$\frac{\partial U^{be}(\vec{E})}{\partial \vec{r}_{i\alpha}} = -\sum_{ib} 2k_{\theta;ib}(\vec{E})(\theta_{ib} - \theta_{e;ib}(\vec{E})) \frac{1}{\sin \theta_{ib}} \frac{\partial \cos \theta_{ib}}{\partial \vec{r}_{i\alpha}} \quad (25)$$

$$\frac{\partial U^{tor}(\vec{E})}{\partial \vec{r}_{i\alpha}} = -\sum_{it} \left(\sum_{i=0}^6 i C_{it;i}(\vec{E}) \cos^{i-1}(\varphi_{it} + \varphi_{0,it}(\vec{E})) \sin(\varphi_{it} + \varphi_{0,it}(\vec{E})) \frac{\partial \varphi_{it}}{\partial \vec{r}_{i\alpha}} \right), \quad (26)$$

$$\frac{\partial U^{imp}(\vec{E})}{\partial \vec{r}_{i\alpha}} = \sum_{im} 2k_{t,im}(\vec{E})(t_{im} - t_{e,im}(\vec{E})) \frac{\partial t_{im}}{\partial \vec{r}_{i\alpha}}. \quad (27)$$

where the remaining derivatives are discussed elsewhere².

The evaluation of $\frac{\partial U^{\text{intra}}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial \vec{r}_{i\alpha}}$ and $\frac{\partial U^{\text{intra}}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial \mathbf{Q}_{i\alpha}}$ is more complicated.

U^{intra} is the field-dependent intramolecular energy that includes contributions from all atoms in the system. Changes in any charge or position will affect the electrostatic

fields on all atoms in the system. Therefore, $\frac{\partial U^{\text{intra}}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial \vec{r}_{i\alpha}}$ and $\frac{\partial U^{\text{intra}}}{\partial \vec{E}} \cdot \frac{\partial \vec{E}}{\partial \mathbf{Q}_{i\alpha}}$

should be written as $\sum_{\beta} \sum_j \frac{\partial U^{\text{intra}}}{\partial \vec{E}_{j\beta}} \cdot \frac{\partial \vec{E}_{j\beta}}{\partial \vec{r}_{i\alpha}}$ and $\sum_{\beta} \sum_j \frac{\partial U^{\text{intra}}}{\partial \vec{E}_{j\beta}} \cdot \frac{\partial \vec{E}_{j\beta}}{\partial \mathbf{Q}_{i\alpha}}$, where β

sums over all molecules and j sums over all atoms in each molecule. The most

straightforward way to calculate these terms is to calculate and save all $\frac{\partial \vec{E}_{j\beta}}{\partial \vec{r}_{i\alpha}}$ and

$\frac{\partial \vec{E}_{j\beta}}{\partial Q_{i\alpha}}$ values. However, the evaluation of these terms is O(N) more time consuming than the Ewald sums [20]. In order to speed up the calculations, we optimized the summations so that they are on the same order of Ewald sums. Specifically, Ewald

summation is first called to calculate the electrostatic fields, and then $\frac{\partial U^{\text{intra}}}{\partial \vec{E}_{j\beta}}$ is

calculated for all atoms. With $\frac{\partial U^{\text{intra}}}{\partial \vec{E}_{j\beta}}$ pre-calculated, the total sum can be converted

into a simpler algorithm that is similar to the Ewald. More details are provided below.

The electrostatic field on atom i of molecule α is

$$\begin{aligned} \vec{E}_{i\alpha} = & \frac{2\pi}{V} \sum_{\beta=1}^M \sum_{j=1}^{N_\beta} \sum_{\bar{k}} \frac{Q_{j\beta}}{k^2} \exp(-\frac{k^2}{4\alpha}) \bar{k} \sin(\bar{k} \bullet \vec{r}_{i\alpha j\beta}) \\ & + \sum_{\beta=1}^M \sum_{j=1}^{N_\beta} (1 - \delta_{ij} \delta_{\alpha\beta}) \sum_{\bar{k}} Q_{j\beta} \frac{\operatorname{erfc}(\alpha r_{i\alpha j\beta}) + \frac{2\alpha r_{i\alpha j\beta}}{\sqrt{\pi}} \exp(-\alpha^2 r_{i\alpha j\beta}^2)}{r_{i\alpha j\beta}^3} \vec{r}_{i\alpha j\beta} \\ & - \sum_{j=1}^{N_\alpha} (1 - \delta'_{ij}) \frac{Q_{j\alpha}}{r_{i\alpha j\alpha}^3} \vec{r}_{i\alpha j\alpha}, \end{aligned} \quad (28)$$

where the last sum removes the contribution from atoms that are separated by less than 4 bonds within the same molecule and δ'_{ij} is zero for atoms that are close (separated by 4 or more bonds), and one otherwise. The derivatives of the x -component of this field are

$$\begin{aligned} \sum_{\beta} \sum_j \frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}} \cdot \frac{\partial E_{x,j\beta}}{\partial x_{i\alpha}} = & \frac{2\pi Q_{i\alpha}}{V} \sum_{\bar{k}} [\frac{1}{k^2} \exp(-\frac{k^2}{4\alpha}) k_x^2 \sum_{\beta} \sum_j (1 - \delta_{ij} \delta_{\alpha\beta}) \frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}}] \cos(\bar{k} \bullet \vec{r}_{i\alpha j\beta}) \\ & + Q_{i\alpha} \sum_{\beta} \sum_j (1 - \delta_{ij} \delta_{\alpha\beta}) \frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}} \frac{\operatorname{erfc}(\alpha r_{i\alpha j\beta}) + \frac{2\alpha r_{i\alpha j\beta}}{\sqrt{\pi}} \exp(-\alpha^2 r_{i\alpha j\beta}^2)}{r_{i\alpha j\beta}^3} \end{aligned}$$

$$\begin{aligned}
& -3Q_{i\alpha} \sum_{\beta} \sum_j (1 - \delta_{ij} \delta_{\alpha\beta}) \frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}} \frac{\text{erfc}(\alpha r_{i\alpha j\beta}) + \frac{2\alpha r_{i\alpha j\beta}}{\sqrt{\pi}} \exp(-\alpha^2 r_{i\alpha j\beta}^2) + \frac{4\alpha^3}{3\sqrt{\pi}} r_{i\alpha j\beta}^3 \exp(-\alpha^2 r_{i\alpha j\beta}^2)}{r_{i\alpha j\beta}^5} x_{i\alpha j\beta}^2 \\
& + Q_{i\alpha} \sum_{\beta} \sum_j \delta_{\alpha\beta} (1 - \delta_{ij}) \frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}} \left(-\frac{1}{r_{i\alpha j\beta}^3} + 3 \frac{1}{r_{i\alpha j\beta}^5} x_{i\alpha j\beta}^2 \right). \quad (29)
\end{aligned}$$

Similarly, the derivative with respect to the y-coordinate is

$$\begin{aligned}
& \sum_{\beta} \sum_j \frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}} \cdot \frac{\partial E_{x,j\beta}}{\partial y_{i\alpha}} = \frac{2\pi Q_{i\alpha}}{V} \sum_k \left[\frac{1}{k^2} \exp\left(-\frac{k^2}{4\alpha}\right) k_x k_y \sum_{\beta} \sum_j (1 - \delta_{ij} \delta_{\alpha\beta}) \frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}} \cos(\vec{k} \bullet \vec{r}_{i\alpha j\beta}) \right] \\
& - 3Q_{i\alpha} \sum_{\beta} \sum_j (1 - \delta_{ij} \delta_{\alpha\beta}) \frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}} \frac{\text{erfc}(\alpha r_{i\alpha j\beta}) + \frac{2\alpha r_{i\alpha j\beta}}{\sqrt{\pi}} \exp(-\alpha^2 r_{i\alpha j\beta}^2) + \frac{4\alpha^3}{3\sqrt{\pi}} r_{i\alpha j\beta}^3 \exp(-\alpha^2 r_{i\alpha j\beta}^2)}{r_{i\alpha j\beta}^5} x_{i\alpha j\beta} y_{i\alpha j\beta} \\
& + 3Q_{i\alpha} \sum_{\beta} \sum_j \delta_{\alpha\beta} (1 - \delta_{ij}) \frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}} \frac{1}{r_{i\alpha j\beta}^5} x_{i\alpha j\beta} y_{i\alpha j\beta}. \quad (30)
\end{aligned}$$

Other derivatives including those with respect to charges can be derived using the same approach. All these summations can be calculated by an algorithm that is very similar to the one used to calculate Ewald sums, and only small modifications are needed to

include the pre-calculated $\frac{\partial U^{\text{intra}}}{\partial E_{x,j\beta}}$ values.

3.2 Nosé-Hoover Thermostats

For MD simulations performed under constant temperature conditions, we recommend using an extended Lagrangian method following Nosé and Hoover's work [21,22]. Similarly, the "temperature" of the fluctuating charges should be maintained by another Nosé-Hoover thermostat. In particular, the charges should be kept "cold" to limit energy transfer from atomic motion degrees of freedom. The Hamiltonian of this extended system is

$$H_{NH} = \sum_{\alpha}^M \sum_i^{N_{\alpha}} \frac{\vec{p}_{i\alpha}^2}{2m_{i\alpha}} + \frac{p_{tr}^2}{2W_{tr}} + g_{tr} k_B T_{tr} \ln s_{tr} + \sum_{\alpha}^M \sum_i^{N_{\alpha}} \frac{p_{Q,i\alpha}^2}{2m_Q} + \frac{p_{fq}^2}{2W_{fq}} + g_{fq} k_B T_{fq} \ln s_{fq} + U(\vec{r}^N, Q^N) \quad (31)$$

where s_{tr} and s_{fq} are the additional variables for atomic motion and charge fluctuation, respectively, W_{tr} and W_{fq} are the effective masses, and p_{tr} and p_{fq} are their

momenta. In Eq. (31), g_{tr} and g_{fq} are the degrees of freedom plus one, and $m_{i\alpha}$ and m_Q are the atomic mass and the fictitious charge mass, respectively. The fictitious mass of the charges must be chosen with care. Specifically, this mass should be small enough so that charge fluctuations can rapidly respond to molecular motions. We have found that larger values of m_Q lead to unphysical self-diffusion coefficients for ethanol [13]. The translational temperature of $T_{tr} = 298\text{K}$ is maintained with the fictitious variables $\eta_{tr} = \ln s_{tr}$ and $p_{tr} = W_{tr}\dot{\eta}_{tr}$ with a mass of W_{tr} . The atomic charge fluctuation is kept at a temperature of $T_{fq} = 1\text{K}$ with the fictitious variables $\eta_{fq} = \ln s_{fq}$ and $p_{fq} = W_{fq}\dot{\eta}_{fq}$.

3.3 Multiple Time Steps

In MD simulations, the charges should fluctuate much faster than nuclear motion since they are implicitly representing electronic motion within the molecule. When the fCINTRA model is applied, the coupling between the rapid changes in the charges (and the electrostatic field) and the intramolecular potential requires a small time step (hundredths of a fs) to yield a conserved value for H_{NH} (Eq. (31)). Evaluation of all the forces for such a small time step is prohibitive and simulations would be restricted to an unreasonably small time frame. To increase the time step for the atomic positions, we introduce a reversible multiple time step algorithm for integrating the equations of motion. The general details of multiple time step algorithms have been discussed elsewhere [13,23]. Here, we will focus on the implementation of the algorithm for polarizable and flexible models.

In a system without charge fluctuations, the reversible propagator can be defined as

$$G(\Delta t) = e^{\frac{\Delta t}{2} F_{\vec{r}}(Q, \vec{r}) \bullet \vec{\nabla}_{\vec{p}}} e^{\Delta t \dot{\vec{r}} \bullet \vec{\nabla}_{\vec{r}}} e^{\frac{\Delta t}{2} F_{\vec{r}}(Q, \vec{r}) \bullet \vec{\nabla}_{\vec{p}}} \quad (32)$$

where the momenta are propagated forward in time, by $\Delta t/2$, followed by a full time step propagation of the positions, and finally another half time step propagation of the momenta. This division can be repeated as many times as required [23] leading to an endless family of propagators of increasing accuracy. We have found that a suitable propagator for polarizable and flexible models is

$$G(\Delta t) = e^{\frac{\Delta t}{2} F_{\vec{r}}(Q, \vec{r}) \bullet \vec{\nabla}_{\vec{p}}} e^{\frac{\Delta t}{2} \dot{\vec{r}} \bullet \vec{\nabla}_{\vec{r}}} \times \left(e^{\frac{\delta t}{2} F_Q(\vec{r}, Q) \frac{\partial}{\partial p_Q}} e^{\delta t \dot{Q} \frac{\partial}{\partial Q}} e^{\frac{\delta t}{2} F_Q(\vec{r}, Q) \frac{\partial}{\partial p_Q}} \right)^M \times e^{\frac{\Delta t}{2} \dot{\vec{r}} \bullet \vec{\nabla}_{\vec{r}}} e^{\frac{\Delta t}{2} F_{\vec{r}}(Q, \vec{r}) \bullet \vec{\nabla}_{\vec{p}}}, \quad (33)$$

where the charge-related operators are placed in the center bracket which will be iterated M times with a smaller time step of $\delta t = \Delta t / M$. In this way, the overall time step Δt is consistent with the time scale for nuclear motion but the charges vary over the smaller time step δt . The propagator in Eq. (33) leads to an important reduction in simulation cost since the atomic positions, and associated CPU intensive force calculations, are minimized.

In our simulation, the fictitious variables from Nosé-Hoover thermostats also vary with time and must be included. The complete, reversible propagator is [13]

$$G(\Delta t) = e^{\frac{\Delta t}{2} F_{\eta_{tr}}(\vec{p}_{i\alpha}) \frac{\partial}{\partial p_{tr}}} \left[e^{\frac{\Delta t}{4} F_{\vec{r}}(Q, \vec{r}) \bullet \vec{\nabla}_{\vec{p}}} e^{-\frac{\Delta t}{2} \dot{\eta}_{tr} \vec{p} \bullet \vec{\nabla}_{\vec{p}}} e^{\frac{\Delta t}{4} F_{\vec{r}}(Q, \vec{r}) \bullet \vec{\nabla}_{\vec{p}}} \right] e^{\frac{\Delta t}{2} \dot{\eta}_{tr} \frac{\partial}{\partial \eta_{tr}}} e^{\frac{\Delta t}{2} \dot{\vec{r}} \bullet \vec{\nabla}_{\vec{r}}} \times \\ \left(e^{\frac{\delta t}{2} F_{\eta_{f_q}}(p) \frac{\partial}{\partial p_{f_q}}} \left[e^{\frac{\delta t}{4} F_Q(\vec{r}, Q) \frac{\partial}{\partial p_Q}} e^{-\frac{\delta t}{2} \dot{\eta}_{f_q} p_Q \frac{\partial}{\partial p_Q}} e^{\frac{\delta t}{4} F_Q(\vec{r}, Q) \frac{\partial}{\partial p_Q}} \right] e^{\frac{\delta t}{2} \dot{\eta}_{f_q} \frac{\partial}{\partial \eta_{f_q}}} e^{\delta t \dot{Q} \frac{\partial}{\partial Q}} e^{\frac{\delta t}{2} \dot{\eta}_{f_q} \frac{\partial}{\partial \eta_{f_q}}} \times \right)^M \right. \\ \left. \left(e^{\frac{\delta t}{4} F_Q(\vec{r}, Q) \frac{\partial}{\partial p_Q}} e^{-\frac{\delta t}{2} \dot{\eta}_{f_q} p_Q \frac{\partial}{\partial p_Q}} e^{\frac{\delta t}{4} F_Q(\vec{r}, Q) \frac{\partial}{\partial p_Q}} \right] e^{\frac{\delta t}{2} F_{\eta_{f_q}}(p) \frac{\partial}{\partial p_{f_q}}} \right. \\ \left. \times e^{\frac{\Delta t}{2} \dot{\vec{r}} \bullet \vec{\nabla}_{\vec{r}}} e^{\frac{\Delta t}{2} \dot{\eta}_{tr} \frac{\partial}{\partial \eta_{tr}}} \left[e^{\frac{\Delta t}{4} F_{\vec{r}}(Q, \vec{r}) \bullet \vec{\nabla}_{\vec{p}}} e^{-\frac{\Delta t}{2} \dot{\eta}_{tr} \vec{p} \bullet \vec{\nabla}_{\vec{p}}} e^{\frac{\Delta t}{4} F_{\vec{r}}(Q, \vec{r}) \bullet \vec{\nabla}_{\vec{p}}} \right] e^{\frac{\Delta t}{2} F_{\eta_{tr}}(\vec{p}_{i\alpha}) \frac{\partial}{\partial p_{tr}}} \right) \right) \quad (34)$$

where the charge-dependent Nosé-Hoover variables appear in the inner loop. For ethanol, the positions were advanced with a time step Δt of 0.1fs, while the charges fluctuated with a time step δt of 0.01fs. With these values, the inner loop was iterated 10 times ($M=10$). For other molecular systems, different time steps may be required but those used for ethanol serve as a guide. In all cases, the time invariance of H_{NH} should be examined to decide on appropriate time steps.

The operators in the propagator act on the system sequentially, from right to left. For

instance, the rightmost term, $e^{\frac{\Delta t}{2} F_{\eta_{tr}}(\vec{p}_{i\alpha}) \frac{\partial}{\partial p_{tr}}}$, acts on the system first and propagates p_{tr} from time t to $t+\Delta t/2$:

$$p_{tr}(t + \Delta t / 2) = p_{tr}(t) + \frac{\Delta t}{2} F_{\eta_{tr}}(\vec{p}_{i\alpha}(t)) . \quad (35)$$

Following the same rules, equations of motion from all other terms can be written out.

All the terms in the first and last lines of Eq. (34) act on atomic positions, momenta, and associated Nosé-Hoover variables. Those in the middle line act on charge-related variables. After the second and third lines of Eq. (34) are applied to the system, the position-related variables are propagated to $t+\Delta t/2$ and the charge-related variables are fully propagated to $t+\Delta t$.

Consider the terms in the first line. First, the rightmost term in the first line, $e^{\frac{\Delta t}{2}\vec{r}\bullet\vec{\nabla}_{\vec{r}}}$, propagates $\vec{r}_{i\alpha}$ from time $t+\Delta t/2$ to $t+\Delta t$:

$$\vec{r}_{i\alpha}(t+\Delta t) = \vec{r}_{i\alpha}(t + \frac{\Delta t}{2}) + \frac{\Delta t}{2m_{i\alpha}} \vec{p}_{i\alpha}(t + \frac{\Delta t}{2}). \quad (36)$$

At this stage, positions and velocities have been advanced to $t+\Delta t$ and $t+\Delta t/2$, respectively. If there are any internal constraints in the molecules, such as fixed bond lengths, constraint algorithms can be applied for the positions here. For instance, the first part of the RATTLE algorithm [24] can be used here to iteratively calculate the positions that meet the constraints. Forces at time $t+\Delta t$ should be calculated at this point. Then

$e^{\frac{\Delta t}{2}\dot{\eta}_{tr}\frac{\partial}{\partial\eta_{tr}}}$ propagates η_{tr} from time $t+\Delta t/2$ to $t+\Delta t$ as follows

$$\eta_{tr}(t+\Delta t) = \eta_{tr}(t + \frac{\Delta t}{2}) + \frac{\Delta t}{2W_{tr}} p_{tr}(t + \frac{\Delta t}{2}) \quad (37)$$

After that, $e^{\frac{\Delta t}{4}F_{\vec{r}}(Q,\vec{r})\bullet\vec{\nabla}_{\vec{p}}}$ propagates $\vec{p}_{i\alpha}$ from time $t + \frac{\Delta t}{2}$ to $t + \frac{3\Delta t}{4}$ as follows

$$\vec{p}_{i\alpha}(t + \frac{3\Delta t}{4}) = \vec{p}_{i\alpha}(t + \frac{\Delta t}{2}) + \frac{\Delta t}{4} F_{\vec{r}}(Q, \vec{r}; t + \Delta t), \quad (38)$$

and $e^{-\frac{\Delta t}{2}\dot{\eta}_{tr}\vec{p}\bullet\vec{\nabla}_{\vec{p}}}$ will result in $\vec{p}_{i\alpha}(t + \frac{3\Delta t}{4})$ being multiplied by $e^{-\frac{\Delta t}{2}\dot{\eta}_{tr}\vec{p}}$

$$\vec{p}_{i\alpha}(t + \frac{3\Delta t}{4}) = [\vec{p}_{i\alpha}(t + \frac{\Delta t}{2}) + \frac{\Delta t}{4} F_{\vec{r}}(Q, \vec{r}; t + \Delta t)] e^{-\frac{\Delta t}{2} p_{tr}(t + \frac{\Delta t}{2}) W_{tr}}, \quad (39)$$

and then, another $e^{\frac{\Delta t}{4}F_{\vec{r}}(Q,\vec{r})\bullet\vec{\nabla}_{\vec{p}}}$ propagates $\vec{p}_{i\alpha}$ from time $t + \frac{3\Delta t}{4}$ to $t + \Delta t$ as

follows

$$\vec{p}_{i\alpha}(t + \Delta t) = \vec{p}_{i\alpha}(t + \frac{3\Delta t}{4}) + \frac{\Delta t}{4} F_{\vec{r}}(Q, \vec{r}; t + \Delta t) . \quad (40)$$

Therefore, in total, $\vec{p}_{i\alpha}$ is propagated from time $t + \frac{\Delta t}{2}$ to $t + \Delta t$:

$$\vec{p}_{i\alpha}(t + \Delta t) = [\vec{p}_{i\alpha}(t + \frac{\Delta t}{2}) + \frac{\Delta t}{4} F_{\vec{r}}(Q, \vec{r}; t + \Delta t)] e^{-\frac{\Delta t}{2} p_{tr}(t + \frac{\Delta t}{2}) W_{tr}} + \frac{\Delta t}{4} F_{\vec{r}}(Q, \vec{r}; t + \Delta t) . \quad (41)$$

After this step, positions and velocities are all fully advanced to $t + \Delta t$, and the second part of the RATTLE algorithm can be applied to calculate the final velocities that fit the constraints.

Finally, $e^{\frac{\Delta t}{2} F_{\eta_{tr}}(\vec{p}_{i\alpha}) \frac{\partial}{\partial p_{tr}}}$ advances p_{tr} from time $t + \Delta t/2$ to $t + \Delta t$:

$$p_{tr}(t + \Delta t) = p_{tr}(t + \frac{\Delta t}{2}) + \frac{\Delta t}{2} F_{\eta_{tr}}(\vec{p}_{i\alpha}(t + \Delta t)) . \quad (42)$$

Therefore, after all terms in Eq. (34) are applied, all variables in the system have advanced by Δt .

3.4 Optimization

The propagation of the system is described by the flow chart in Fig.3. In each time step, forces are calculated at three places: before the inner loop, in the inner loop, and before the end of the whole time step. The first two force calculations are for charge fluctuations (Eq. (19)), and the last one is for atomic positions (Eq. (18)). All three force calculations are somewhat similar, but the second one is inside the inner loop and should be made as fast as possible. Specifically, we save the results of the most time-consuming evaluations from the first force calculation, such as the evaluation of exponentials, and re-use them in the inner loop. This saves calculation time by more than 80% in the inner loop.

In the force evaluation, we use Message Passing Interface (MPI) to parallelize the calculations. Some implementations [2,25-27] of the parallel MD simulations subdivide the simulation cell into smaller cells (the Linked-Cell method [2]), with a processor assigned to each cell. Others employ neighbor lists [26]. We subdivide the computation between processors using a different algorithm. At the very beginning of the simulation, the total number of atomic pairs is subdivided according to the number of processors and this division is maintained throughout the simulation. Since we employ the largest possible spatial cutoff for Lennard-Jones interactions, neighbor lists and Linked-Cells are not appropriate. Specifically, within the simulations, a very large number of molecular pairs would be counted as neighbors. In our implementation, the processors only communicate before and after the force calculations. Also, there is no

overhead required to update lists... as would be required in a neighbor list or Linked-Cell approach. Our method minimizes the total number of communication calls, eliminates any CPU time needed to update lists, and does not compromise accuracy with a small cutoff distance and infrequent list updates.

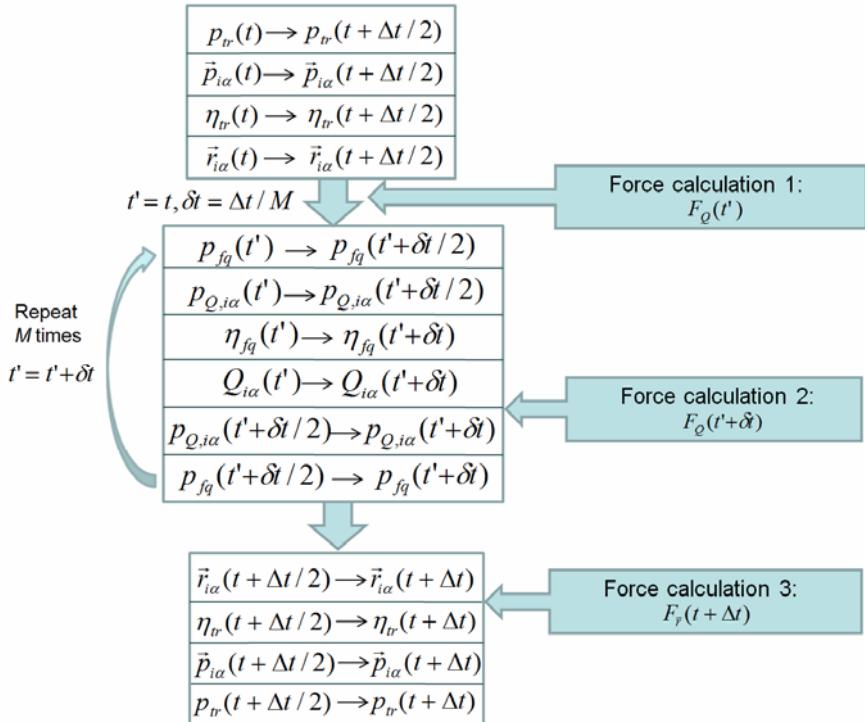


Fig. 3. Flow chart showing the propagation of the system

In MD simulations the “costs” of various tasks can be assigned a definite dependence on N , the number of atoms in the system. We have parallelized all calculations that require more than $\mathcal{O}(N)$ but, for very large systems, additional parallelization could prove useful. It is important to realize that, in parallelizing an MD program, the number of communication calls between nodes is independent of N . In this way, the communication time should not increase significantly while the computational time will increase as N^2 .

Our parallelization procedure is as follows: First of all, the work is evenly divided among the processors, and the necessary information, such as the atomic charges and positions, is broadcast to all processors. Then each processor does part of the calculation and, after all calculations are done, the results are summed up on the master node

(a)

```
C* Original: Done at every time step
    do 10 i=1,npert
        do 20 ia=1,natom
            do 30 j=i,npert
                do 40 jb=1,natom
                    if((i.ne.j).or.(i.eq.j.and.jb.gt.ia))then
                        if(DIST(i,ia,j,jb).lt.rcut) then
                            CALCULATIONS
                        endif
                    endif
                40      continue
                30      continue
                20      continue
            10      continue
```

(b)

```
C* Revised:
C* Preliminary step. Done only once at the beginning
C* of simulation.
C* Prepare index that contains information for
C* atom pairs. molmax is the maximum number of
C* molecules in the system and maxsite is the
C* maximum number of atoms in each molecule.
C* myrank is the index of the processor.
    nsquared=molmax**2
    ncubed=maxsite*molmax**2
    npair=0
    do 10 i=1,npert
        do 20 ia=1,natom
            do 30 j=i,npert
                do 40 jb=1,natom
                    if((i.ne.j).or.(i.eq.j.and.jb.gt.ia))then
                        npair=npair+1
                        masterind(npair)=i+j*(molmax)+1
                        ia*nsquared+jb*ncubed
                    endif
                40      continue
                30      continue
                20      continue
            10      continue

C* Then evenly divide masterind to get indexStart
C* and indexEnd for all processors.
C* nprocs is the number of processors.
    npairProc=npair/nprocs
    npairProcMax=npairProc+1
    npairResidual=npair-npairProc*nprocs
    do i=0,npairResidual-1
        npairBegin(i)=npairProcMax*i + 1
        npairEnd(i)=npairProcMax*(i+1)
    enddo
    do i=npairResidual,nprocs-1
        npairBegin(i)=npairProcMax*npairResidual+
    1        npairProc*(i-npairResidual)+1
        npairEnd(i)=npairProcMax*npairResidual+
    1        npairProc*(i-npairResidual+1)
    enddo

    indexStart=npairBegin(myrank)
    indexEnd=npairEnd(myrank)
```

Fig. 4. Fortran code showing the parallelization of the force calculations. (a) is the original serial code. (b) is the preliminary part for parallel calculations and is done only once at the beginning of the simulation. (c) is the revised code for parallelized force calculations.

```
(c)
```

```

C* Revised: Done at every time step.
      do 10 k=indexStart, indexEnd
C*   Unpacking i,ia,j,jb
      indmaster=masterind(k)
      jb=int(indmaster/ncubed)
      india=mod(indmaster,ncubed)
      ia=int(india/nsquared)
      indj=mod(india,nsquared)
      j=int(indj/molmax)
      i=mod(indj,molmax)

      if(DIST(i,ia,j,jb).lt.rcut) then
         CALCULATIONS
      endif
      continue
10

```

Fig. 4. (continued)

and broadcast to all processors so that each processor has a full copy of the results. An example of an $\mathcal{O}(N^2)$ calculation, before and after parallelization, is given in Fig.4. The four loop structure, with embedded if statement, has been replaced by a single loop with unpacking of an index to retrieve the four original variables. The range of the single loop is trivially divided among the processors so that the processor load is very well shared.

3.5 Ethanol: An Example

Test simulations of bulk ethanol were run on the High Performance Computing Virtual Laboratory (HPCVL) Sun Fire 25000 clusters, each of which has 72 dual-core UltraSPARC-IV+ 1.5 GHz processors and response crossbar interconnect. The scaling of the program is given in Fig.5. The program scales very well and, as noted above, scaling is better for larger systems (500 molecules) than smaller systems (256 molecules). It can be inferred that for even larger systems, such as 1000 molecules, the scaling factor will be further improved. Compared to non-polarizable models, the fCINTRA model is slower because of the more detailed molecular representation, the inevitably more complicated force calculations, and the requirement for multiple time steps. A test on HPCVL with 8 processors shows that the simulation time for 1000 time steps with a non-polarizable OPLS model [19], the FC model where only the charges fluctuate in time, and the fCINTRA model where the charges and the intramolecular potential vary dynamically, are 32.25, 35.58 and 337.3 seconds, respectively. All the simulations are performed using the MDMC program [28]. Overall, the fCINTRA model is about 10.5 times slower than the non-polarizable model – acceptable given the improved molecular description.

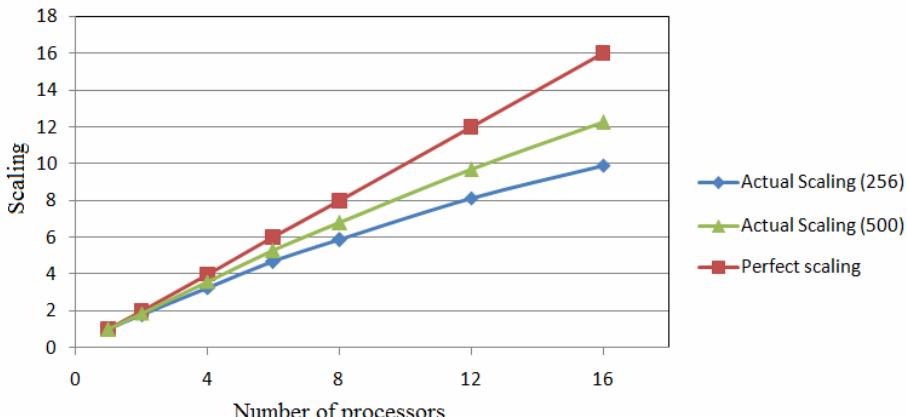


Fig.5. Scaling of the simulation of 256 and 500 ethanol molecules based on the fCINTRA model. Results are presented for computations on a Sunfire 25K.

4 Conclusions

In this article, details on the implementation of polarizable and flexible molecular models (fCINTRA) for MD simulations are described. The force calculations for the fCINTRA model are much more complicated than those for non-polarizable models, scaling in principle as N^3 , but they are significantly simplified by careful code optimization and the scaling is reduced to N^2 . The multiple time step algorithm successfully separates the charge fluctuations and atomic motions and greatly increases the simulation efficiency. The implementation of the multiple time step algorithm is described in detail and the methods to improve the efficiency are working well. The program scales well and will scale even better for larger systems.

Acknowledgements. The financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) is gratefully acknowledged. S. Wang thanks the High Performance Computing Virtual Lab (HPCVL) for the award of a Sun Microsystems of Canada Scholarship. The computational resources at HPCVL, SHARCNET and Westgrid are gratefully acknowledged.

References

1. Mackerell, A.D.: Empirical force fields for biological macromolecules: Overview and issues. *Journal of Computational Chemistry* 25(13), 1584–1604 (2004)
2. Allen, M.P., Tildesley, D.J.: Computer Simulation of Liquids, 1st edn. Oxford University Press, New York (1987)
3. Shirts, M.R., Pitera, J.W., Swope, W.C., Pande, V.S.: Extremely precise free energy calculations of amino acid side chain analogs: Comparison of common molecular mechanics force fields for proteins. *Journal of Chemical Physics* 119(11), 5740–5761 (2003)

4. Car, R., Parrinello, M.: Unified Approach for Molecular-Dynamics and Density-Functional Theory. *Physical Review Letters* 55(22), 2471–2474 (1985)
5. Spohr, E.: Some recent trends in computer simulations of aqueous double layers. *Electrochimica Acta*, 23–27 (2003)
6. Gao, J.L., Habibollazadeh, D., Shao, L.: A polarizable intermolecular potential function for simulation of liquid alcohols. *Journal of Physical Chemistry* 99(44), 16460–16467 (1995)
7. Lamoureux, G., MacKerell, A.D., Roux, B.: A simple polarizable model of water based on classical Drude oscillators. *Journal of Chemical Physics* 119(10), 5185–5197 (2003)
8. Lamoureux, G., Roux, B.: Modeling induced polarization with classical Drude oscillators: Theory and molecular dynamics simulation algorithm. *Journal of Chemical Physics* 119(6), 3025–3039 (2003)
9. Noskov, S.Y., Lamoureux, G., Roux, B.: Molecular dynamics study of hydration in ethanol-water mixtures using a polarizable force field. *Journal of Physical Chemistry B* 109(14), 6705–6713 (2005)
10. Rick, S.W., Stuart, S.J., Berne, B.J.: Dynamical fluctuating charge force-fields - Application to liquid water. *Journal of Chemical Physics* 101(7), 6141–6156 (1994)
11. Svishchev, I.M., Kusalik, P.G., Wang, J., Boyd, R.J.: Polarizable point-charge model for water: Results under normal and extreme conditions. *Journal of Chemical Physics* 105(11), 4742–4750 (1996)
12. Cicu, P., Demontis, P., Spanu, S., Suffritti, G.B., Tilocca, A.: Electric-field-dependent empirical potentials for molecules and crystals: A first application to flexible water molecule adsorbed in zeolites. *Journal of Chemical Physics* 112(19), 8267–8278 (2000)
13. Wang, S.H., Cann, N.M.: Polarizable and flexible model for ethanol. *Journal of Chemical Physics* 126(21), 214502 (2007)
14. Politzer, P., Boyd, S.: Molecular dynamics simulations of energetic solids. *Structural Chemistry* 13(2), 105–113 (2002)
15. Weiner, S.J., Kollman, P.A., Case, D.A., Singh, U.C., Ghio, C., Alagona, G., Profeta, S., Weiner, P.: A New Force-Field for Molecular Mechanical Simulation of Nucleic-Acids and Proteins. *Journal of the American Chemical Society* 106(3), 765–784 (1984)
16. Ryckaert, J.P., Bellemans, A.: Molecular-Dynamics of Liquid Normal-Butane Near its Boiling-Point. *Chemical Physics Letters* 30(1), 123–125 (1975)
17. Rappe, A.K., Goddard, W.A.: Charge Equilibration for Molecular-Dynamics Simulations. *Journal of Physical Chemistry* 95(8), 3358–3363 (1991)
18. Patel, S., Brooks, C.L.: CHARMM fluctuating charge force field for proteins: I parameterization and application to bulk organic liquid simulations. *Journal of Computational Chemistry* 25(1), 1–15 (2004)
19. Jorgensen, W.L.: Optimized Lintermolecular Potential Functions for Liquid Alcohols. *Journal of Physical Chemistry* 90(7), 1276–1284 (1986)
20. Yeh, I.C., Berkowitz, M.L.: Ewald summation for systems with slab geometry. *Journal of Chemical Physics* 111(7), 3155–3162 (1999)
21. Hoover, W.G.: Canonical dynamics – equilibrium phase-space distributions. *Physical Review A* 31(3), 1695–1697 (1985)
22. Nose, S.: A unified formulation of the constant temperature molecular-dynamics methods. *Journal of Chemical Physics* 81(1), 511–519 (1984)
23. Tuckerman, M., Berne, B.J., Martyna, G.J.: Reversible multiple time scale molecular-dynamics. *Journal of Chemical Physics* 97(3), 1990–2001 (1992)
24. Andersen, H.C.: Rattle- A Velocity Version of the Shake Algorithm for Molecular-Dynamics Calculations. *Journal of Computational Physics* 52(1), 24–34 (1983)

25. Brown, D., Clarke, J.H.R., Okuda, M., Yamazaki, T.: A Domain Decomposition Parallelization Strategy for Molecular-Dynamics Simulations on Distributed Memory Machines. *Computer Physics Communications* 74(1), 67–80 (1993)
26. Mason, D.R.: Faster neighbour list generation using a novel lattice vector representation. *Computer Physics Communications* 170(1), 31–41 (2005)
27. Rapaport, D.C.: Large-Scale Molecular-Dynamics Simulation Using Vector and Parallel Computers. *Computer Physics Reports* 9(1), 1–53 (1988)
28. Cressman, E., Das, B., Dunford, J., Pecheanu, R., Huh, Y., Nita, S., Paci, I., Zhao, C., Wang, S., Cann, N.M.: MCMD program (unpublished)

Efficient Numerical Methods for the Time-Dependent Schroedinger Equation for Molecules in Intense Laser Fields

André D. Bandrauk* and HuiZhong Lu**

Laboratoire de Chimie Théorique,
Université de Sherbrooke,
Sherbrooke, Que, J1K 2R1, Canada

Abstract. We present an accurate method for the nonperturbative numerical solution of the Time-Dependent Schroedinger Equation, TDSE, for molecules in intense laser fields, using cylindrical/polar coordinates systems. For cylindrical coordinates systems, after use of a split-operator method which separates the z direction propagation and the (x, y) plane propagation, we approximate the wave function in each (x, y) section by a Fourier series ($\sum c_m(\rho)e^{im\phi}$) which offers an exponential convergence in the ϕ direction and is naturally applicable for polar coordinates systems. The coefficients ($c_m(\rho)$) are then calculated by a Finite Difference Method (FDM) in the ρ direction. We adopt the Crank-Nicholson method for the temporal propagation. The final linear system consists of a set of independent one-dimensional (ρ) linear systems and the matrix for every one-dimensional linear systems is *sparse*, so the whole linear system may be very efficiently solved. We note that the Laplacean operator in polar/cylindrical coordinate has a singular term near the origin. We present a method to improve the numerical stability of the Crank-Nicholson method in this case. We illustrate the improved stability by calculating several eigenstates of H_3^{++} with propagation in imaginary time. Two methods of spatial discretization are also compared in calculations of Molecular High Order Harmonic Generation, MHOHG.

1 Introduction

The current advent of ultrashort intense laser pulses [1] necessitates the development of nonperturbative numerical methods to solve problems of interaction of lasers with molecules [2]. Because of efficiency, in many cases one uses cylindrical coordinates for the numerical solution of the 3-D Time Dependent Schroedinger Equation, TDSE, for linear molecules exposed parallel to an intense laser pulse: H_2^+ [3, 6, 8, 9], H_2 [10]. Generally, we use the split-operator method or the ADI (Alternating Direction Implicit Method [11]) method which separate the propagation in the different directions. This process reduces a lot of computing time

* Canada Research Chair in Computational Chemistry and Photonics.

** RQCHP, Université de Sherbrooke.

and is very precise [11]. After splitting, one must propagate the wave function in each section plane in the polar coordinate. Polar coordinates have also been used in 2-D simulations of molecular alignment [8]. But in the polar coordinate system, $\rho = \sqrt{x^2 + y^2}$, z , one encounters often the unitarity precision problem in the ρ direction when one uses Finite Difference Methods (FDM) [11] and the difficulty of very expensive algorithms in CPU-time when one uses a development based on special functions such as Bessel functions [3].

To overcome the preceding difficulties of FDM in the ρ direction, our idea is to first write the Laplacean operator $\frac{\partial^2}{\partial\rho^2} + \frac{1}{\rho}\frac{\partial}{\partial\rho}$ in the form $-\frac{1}{\rho}\frac{\partial}{\partial\rho}\rho\frac{\partial}{\partial\rho}$ which is self-adjoint with respect to the volume element $\rho d\rho d\phi$ and this helps us to obtain a unitary discrete operator (for detail, see next section). To avoid the problem of boundary conditions at $\rho = 0$, we propagate the wave function in the whole plane without separating the usual cylindrical coordinates ρ and ϕ .

2 The Numerical Method

In the present study, we assume the molecule is parallel to the field [3]. This is justified by the fact that at high intensities molecules align with the field since parallel transition moments are strongest in symmetric molecules. Alignment further corresponds to rotational wavepackets coherently acting to align a molecule. Thus using a model of a molecule parallel to the laser polarization satisfies the above conditions of alignment by superposition of many rotational states.

We present the method for numerical solution of the Time-Dependent Schroedinger Equation in polar coordinates (2D):

$$i\frac{\partial\psi}{\partial t} = -\frac{1}{2} \left[\frac{1}{\rho}\frac{\partial}{\partial\rho}\rho\frac{\partial}{\partial\rho} + \frac{1}{\rho^2}\frac{\partial^2}{\partial\phi^2} \right] \psi + [V_c + \mathbf{r} \cdot \mathbf{E}(t)] \psi \quad (1)$$

$$(0 \leq \rho \leq \rho_{max}, 0 \leq \phi \leq 2\pi) .$$

It is customary to remove the singular $\frac{1}{\rho}\frac{\partial}{\partial\rho}$ term in the Laplacian by defining new function $\rho^{-1/2}\psi$. This introduces new ρ^{-2} singular potentials. We derive here a numerical procedure to circumvent this problem.

We can apply the splitting method to approach the wave propagation in time:

$$\begin{aligned} \psi(t + \delta t) &= \exp[-i(V_c + \mathbf{r} \cdot \mathbf{E}(t)) * \delta t / 2] \\ &\quad * \exp \left[\frac{i}{2} \left(\frac{1}{\rho}\frac{\partial}{\partial\rho}\rho\frac{\partial}{\partial\rho} + \frac{1}{\rho^2}\frac{\partial^2}{\partial\phi^2} \right) * \delta t \right] \\ &\quad * \exp[-i(V_c + \mathbf{r} \cdot \mathbf{E}(t)) * \delta t / 2] \psi(t) + \mathcal{O}(\delta t^3) . \end{aligned} \quad (2)$$

For details on the splitting method we refer to [12], and we concentrate here on studying the second operator of the preceding equation — approximation of the free propagation equation in 2-D. We adopt the Crank-Nicholson method [5] for the temporal approximation of this term:

$$\exp \left[\frac{i}{2} \left(\frac{1}{\rho}\frac{\partial}{\partial\rho}\rho\frac{\partial}{\partial\rho} + \frac{1}{\rho^2}\frac{\partial^2}{\partial\phi^2} \right) * \delta t \right] \psi(\rho, \phi, t + \delta t) \quad (3)$$

$$= \frac{\left[1 + i(\delta t/4) \left(\frac{1}{\rho} \frac{\partial}{\partial \rho} \rho \frac{\partial}{\partial \rho} + \frac{1}{\rho^2} \frac{\partial^2}{\partial \phi^2}\right)\right]}{\left[1 - i(\delta t/4) \left(\frac{1}{\rho} \frac{\partial}{\partial \rho} \rho \frac{\partial}{\partial \rho} + \frac{1}{\rho^2} \frac{\partial^2}{\partial \phi^2}\right)\right]} \psi(\rho, \phi, t) + \mathcal{O}(\delta t^3).$$

For the spatial approximation of (3), we use the truncated Fourier series, [7],

$$\psi(\rho, \phi, t_l) = \frac{1}{N} \sum_{n=-N/2}^{N/2-1} c_n^l(\rho) e^{in\phi}, \quad (4)$$

where $\{c_n^l(\rho)\}$ is the unknown set of functions on ρ . The preceding formula shows that for fixed ρ , $\{c_n^l(\rho)\}_{n=-N/2}^{N/2-1}$ is just the discrete Fourier coefficients of $\psi(\rho, \phi, t_l)$ as a function of ϕ . When ψ is regular in Cartesian coordinates, it is easy to show [7] that we have the polynomial development:

$$c_n(\rho) = \sum_{m>=0} a_{2m} \rho^{|n|+2*m} \text{ for } n = -N/2, \dots, N/2 - 1, \quad (5)$$

where c_n notes for every c_n^l for simplifying notation. It is clear that near the origin, the higher frequency term coefficient function $c_n(\rho)$ is very smooth because it does not contain lower degree polynomial terms of ρ and so we have:

$$\begin{aligned} \frac{d^k c_n}{d\rho^k}(\rho = 0) &= \left[\rho^{|n|-k} \sum_{m>=0} \frac{(|n|+2*m)!}{(|n|+2*m-k)!} a_{2m} \rho^{2*m} \right] (\rho = 0) \\ &= 0 \text{ for } |n| > k. \end{aligned} \quad (6)$$

All coefficient functions $c_n(\rho)$ which don't satisfy the condition (6) are called parasite modes and a function which contains parasite modes is not a regular function in cartesian coordinates. In our representation of the function, we have not insured that our discrete wave functions satisfy the condition (6) so our discrete wavefunction space may have parasite modes.

Now using (4) in (3), we get the following set of ODE's for $c_n^{l+1}(\rho)$:

$$\left[1 - i \frac{\delta t}{4} D_{\rho, n}\right] c_n^{l+1} = \left[1 + i \frac{\delta t}{4} D_{\rho, n}\right] c_n^l \quad (7)$$

for $-N/2 \leq n < N/2$ with the differential operator:

$$D_{\rho, n} = \frac{1}{\rho} \frac{\partial}{\partial \rho} \rho \frac{\partial}{\partial \rho} - \frac{n^2}{\rho^2}. \quad (8)$$

We study two classes of FDM to approximate $c_n(\rho)$ at different sets of ρ points:

$$\text{non-staggered mesh: } \rho_m = m * h \quad \text{for } m \geq 0, \quad (9)$$

$$\text{staggered mesh: } \rho_m = (m + \frac{1}{2}) * h \quad \text{for } m \geq 0. \quad (10)$$

The staggered mesh avoids the Coulomb singularities. For both cases, we adopt the 4th order scheme to discretize the differential part of $D_{\rho, n}$:

$$\frac{1}{\rho} \frac{\partial}{\partial \rho} \rho \frac{\partial}{\partial \rho} u(\rho)$$

$$\begin{aligned}
&= \frac{4}{3} \frac{1}{\rho} \frac{(\rho + h/2) [u(\rho + h) - u(\rho)] - (\rho - h/2) [u(\rho) - u(\rho - h)]}{h * h} \\
&\quad - \frac{1}{3} \frac{1}{\rho} \frac{(\rho + h) [u(\rho + 2h) - u(\rho)] - (\rho - h) [u(\rho) - u(\rho - 2h)]}{2h * 2h} + \mathcal{O}(h^4) .
\end{aligned} \tag{11}$$

At the origin (for non-staggered mesh) for $c_0 = \sum_{m \geq 0} \rho^{2m}$, due to the Coulomb singularity, we use the following approximation:

$$\begin{aligned}
&\frac{1}{\rho} \frac{\partial}{\partial \rho} \rho \frac{\partial}{\partial \rho} c_0 \\
&= 4 \left[-\frac{1}{12h^2} c_0(2h) + \frac{4}{3h^2} c_0(h) - \frac{5}{4h^2} c_0(0) \right] + \mathcal{O}(h^4)
\end{aligned} \tag{12}$$

From the split propagator scheme (23), we see that when the initial condition $\psi(\rho, \phi, t = 0)$ is regular in Cartesian coordinates (satisfying 6), the only way to introduce a parasite mode in our discrete wavefunction is from the Crank-Nicholson method + FDM (7—12). In order to minimize this error, we evaluate the exponential Laplacian by a more general Cranck-Nicholson scheme,

$$\begin{aligned}
&\exp \left[\frac{i}{2} \left(\frac{1}{\rho} \frac{\partial}{\partial \rho} \rho \frac{\partial}{\partial \rho} + \frac{1}{\rho^2} \frac{\partial^2}{\partial \phi^2} \right) * \delta t \right] \psi(\rho, \phi, t + \delta t) \\
&= \left\{ \frac{\left[1 + i(\delta t/4/N_t) \left(\frac{1}{\rho} \frac{\partial}{\partial \rho} \rho \frac{\partial}{\partial \rho} + \frac{1}{\rho^2} \frac{\partial^2}{\partial \phi^2} \right) \right]}{\left[1 - i(\delta t/4/N_t) \left(\frac{1}{\rho} \frac{\partial}{\partial \rho} \rho \frac{\partial}{\partial \rho} + \frac{1}{\rho^2} \frac{\partial^2}{\partial \phi^2} \right) \right]} \right\}^{N_t} \psi(\rho, \phi, t) + \mathcal{O}((\delta t/N_t)^3)
\end{aligned} \tag{13}$$

with $N_t >= 1$. Numerically, the obtained matrix for the Cranck-Nicholson scheme is tridiagonal and the corresponding matrix-vector operation and linear system solution is very fast with Lapack library subroutines.

3 Numerical Results

First, we try to check the precision of our method by comparing the calculated field free eigenstates from this method in polar coordinate and that of a FFT based split-operator method in Cartesian coordinates. The eigenstates of H_3^{++} are calculated by propagating a set of discrete random initialized wavefunctions in imaginary time and conserving their orthonormality between them. The convergence is obtained when $\max |\psi_k^{l+1} - \psi_k^l|/\delta t < 10^{-5}$ is satisfied. Here the Coulomb potential in 2D is regularized to $V_c(r) = \sum_{p=1}^3 \frac{1}{\sqrt{|r-r_p|^2 + 0.18}}$ and the protons' positions are shown in Fig. 1. We list the calculated energies at different level in tables 1–4 for H_3^{++} with different proton position configurations. The discretization parameters are: $\rho_{max} = 64au$, $h = 0.25au$, $N = 128$. Here h is chosen larger than the soft core parameter 0.18 and for the second case with $R = 0.25au = h$, so we have a serious case to examine the accuracy of our numerical method. The results show that the equilateral configuration is easier to calculate than the linear case because the eigen states are more regular

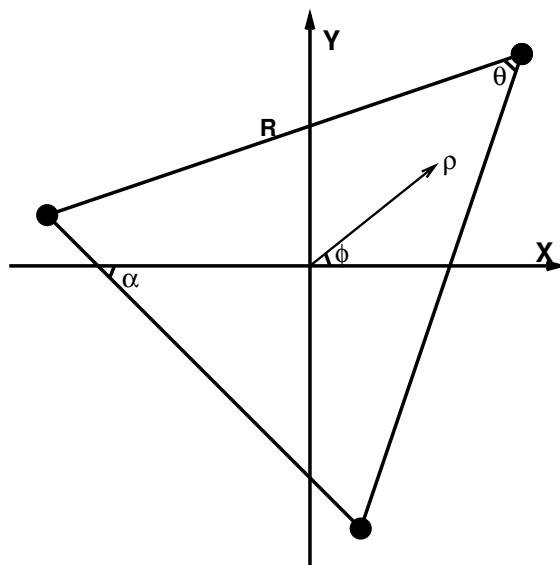


Fig. 1. Proton positions (\bullet) and coordinates for H_3^{++}

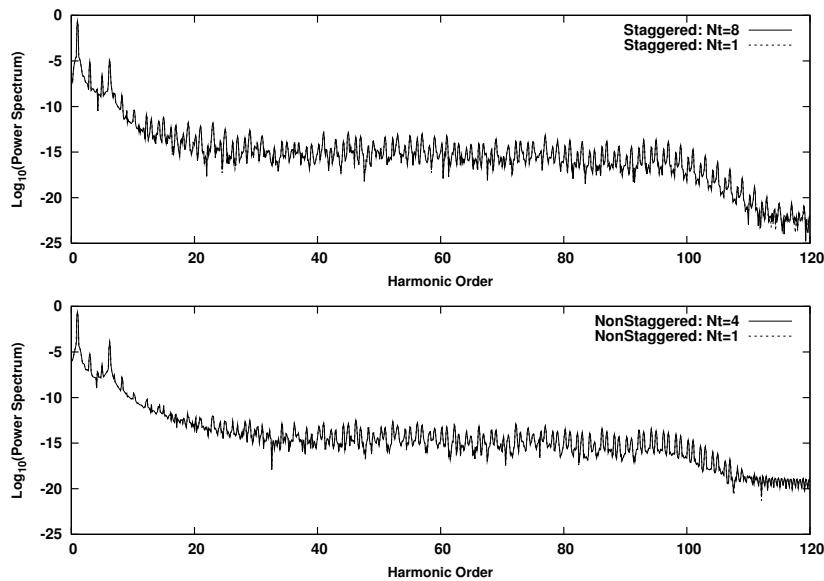


Fig. 2. MHOHG for H_3^{++} ($R=2\text{au}$, Linear), $I=5 \times 10^{14} W/cm^2$, $\lambda = 800nm$ for 2 polar meshes (9,10)

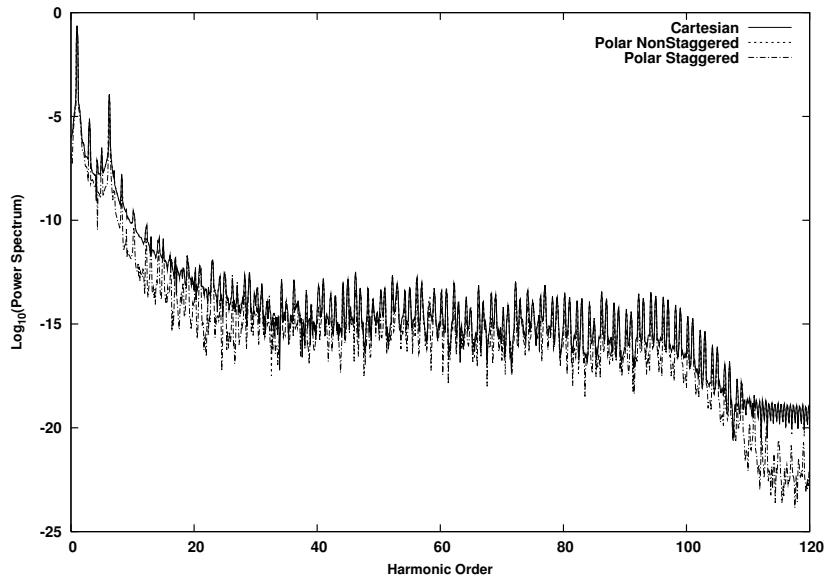


Fig. 3. MHOHG for H_3^{++} ($R=2\text{au}$, Linear) , $I=5 \times 10^{14} W/cm^2$, $\lambda = 800nm$ with 2 polar meshes (9|10) and FFT-Cartesian method

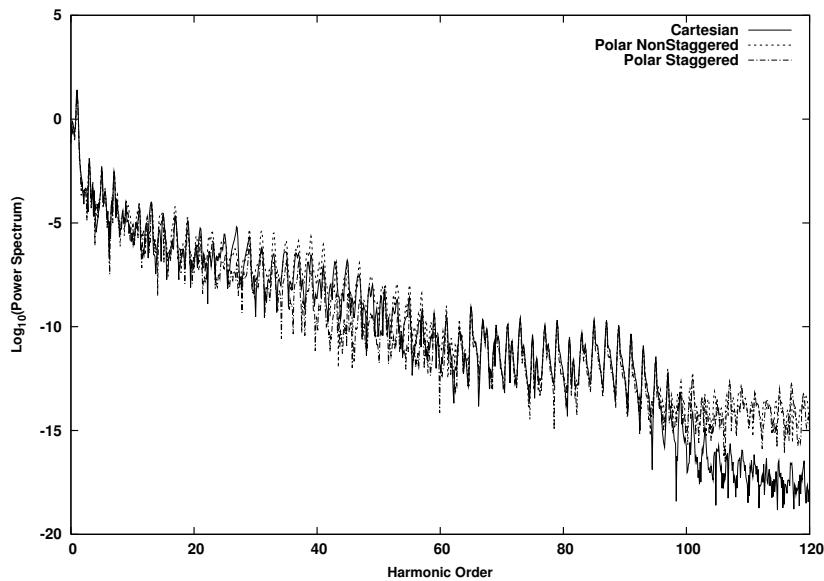


Fig. 4. MHOHG for H_3^{++} ($R=4\text{au}$, Equilateral) , $I=5 \times 10^{14} W/cm^2$, $\lambda = 800nm$ for 2 polar meshes (9|10) and FFT-Cartesian method

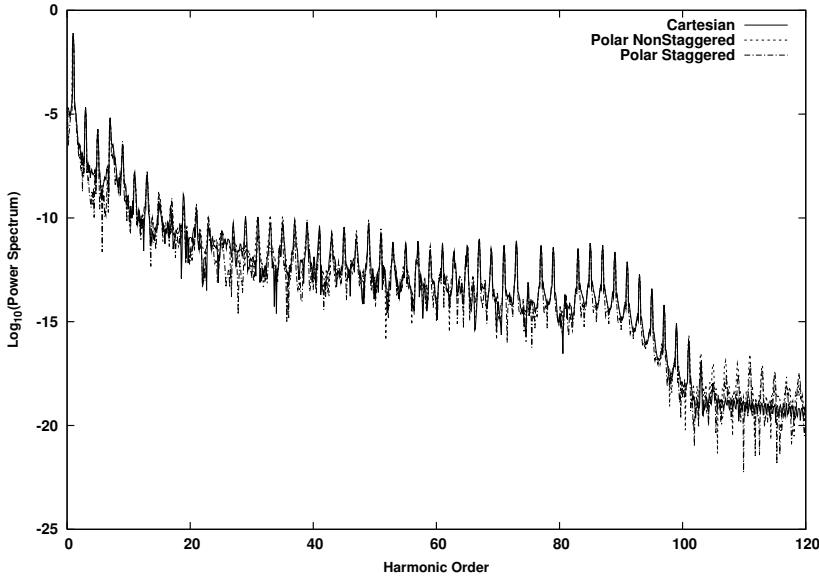


Fig. 5. MHOHG for H_2^+ ($R=2\text{au}$) , $I=5 \times 10^{14} W/cm^2$, $\lambda = 800\text{nm}$ for 2 polar meshes (II) and FFT-Cartesian method (I)

Table 1. Eigenstates energies for H_3^{++} with $R=10$ au, $\theta = \pi/3$ (the equilateral geometry) and $\delta t = 0.02\text{au}$

EL	Cartesian	Pol-NS($N_t = 4$)	Pol-S($N_t = 4$)
0	-0.916947474421911	-0.917000245056269	-0.916991753312918
1	-0.916527337025181	-0.916588252846219	-0.916582907855771
2	-0.916531786622851	-0.916575497255412	-0.916563944321653
3	-0.481162688938491	-0.481168896738671	-0.481168955321553
4	-0.430864364179854	-0.430867615174601	-0.430867710258253
5	-0.430864407222641	-0.430867622132449	-0.430867672957807
6	-0.388363828739741	-0.388373452006733	-0.388335536870061
7	-0.388362676851338	-0.388368320735606	-0.388366088425840
8	-0.388226346779416	-0.388231636070456	-0.388272668508279

EL – Energy Level, Pol-NS – Polar Non-Staggered (II), Pol-S – Polar Staggered (I).

at the origin; the non-staggered method is more efficient when R is small or the protons are in a linear position. In our numerical tests, we noted that we need to use $N_t > 1$ in (II) to obtain accurate numerical results for many cases. Both methods (when $N_t = 8$) are able to give accurately all the first 9 state/energies even with degeneracies.

Secondly, we use our numerical method for MHOHG, Molecular High Order Harmonic Generation [13], real simulations of H_2^+/H_3^{++} 's interaction with an

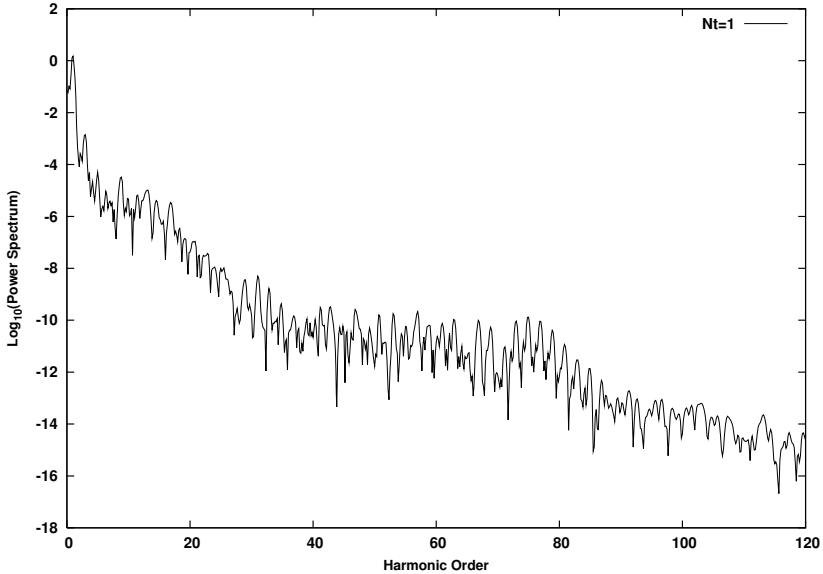


Fig. 6. MHOHG for H_2 ($R=1.675\text{au}$) , $I=5\times 10^{14} W/cm^2$, $\lambda = 800\text{nm}$, duration 6 cycles with the non-staggered polar mesh (9)

Table 2. Eigenstates energies for H_3^{++} with $R=10 \text{ au}$, $\theta = \pi$ (the linear geometry) and $\delta t = 0.02\text{au}$

EL	Cartesian	Pol-NS($N_t = 4$)	Pol-S($N_t = 4$)	Pol-S($N_t = 8$)
0	-0.916207753297108	-0.917263236858678	-1.23846980604331	-0.917795580281749
1	-0.866147930403474	-0.868029817867567	-1.19937639569446	-0.868015103486170
2	-0.866148554807373	-0.868028917449787	-1.19935326798417	-0.868015532725360
3	-0.429565220557124	-0.429596962651704	-1.15835284612339	-0.429591237254770
4	-0.402325399665534	-0.402343557721320	-1.15835284687127	-0.402334900934951
5	-0.394967293543790	-0.395067128340279	-1.11530781393004	-0.395089607240076
6	-0.357048482830901	-0.356968119258126	-1.11530781393008	-0.356968249549419
7	-0.354249155641306	-0.354165720617786	-1.07009310766258	-0.354165367644697
8	-0.350012221814364	-0.350059841036127	-1.07009310766258	-0.350056516994620

EL – Energy Level, Pol-NS – Polar Non-Staggered (9), Pol-S – Polar Staggered (10).

intense laser field: intensity $I = 5\times 10^{14} W/cm^2$, wave length $\lambda = 800\text{nm}$ turned on linearly in 2 cycles with the duration of 10 cycles. The Coulomb potential coefficient is 0.35. The numerical simulation domain is defined as:

1. Cartesian: $|x| \leq 256\text{au}$ and $|y| \leq 256\text{au}$; $\delta t = 0.03\text{au}$; $\delta x = \delta y = 0.25\text{au}$;
2. Polar: $|\rho| \leq 256\text{au}$; $\delta t = 0.03\text{au}$; $\delta \rho = 0.25\text{au}$, $\delta \phi = 2\pi/128$.

where we use atomic units: 1 au (x,y, ρ) = 0.0529 nm, 1 au (t) = $24 \times 10^{-18}\text{s}$. These domain limits $|x|_{max}, |y|_{max}, \rho_{max}$ exceed the maximum classical excursions.

Table 3. Eigenstates energies for H_3^{++} with $R=0.25$ au, $\theta = \pi/3$ (the equilateral geometry) and $\delta t = 0.02au$

EL	Cartesian	Pol-NS($N_t = 4$)	Pol-NS($N_t = 8$)	Pol-S($N_t = 4$)	Pol-S($N_t = 8$)
0	-3.2746833747	-3.2833307815	-3.2833547815	-5.2599857002	-3.2864951188
1	-1.4591773995	-1.4602766909	-1.4602890087	-5.2209035477	-1.5930470062
2	-1.4591780119	-1.4602766909	-1.4602890087	-5.2209035477	-1.4597874732
3	-1.0022509275	-1.0449715583	-1.0050340724	-5.1798209218	-1.4597874732
4	-0.6761159446	-1.0050207127	-0.6761432520	-5.1798209218	-1.4366283126
5	-0.6761160173	-0.8887766581	-0.6761432521	-5.1367876545	-1.4366283126
6	-0.5994537396	-0.8887766581	-0.5999077577	-5.1368098288	-1.2725576542
7	-0.5994537914	-0.7245448911	-0.5999077575	-5.0915840845	-1.2725576542
8	-0.4668429465	-0.7245448911	-0.4678958089	-5.0915840845	-1.1003751893

EL – Energy Level, Pol-NS – Polar Non-Staggered (9), Pol-S – Polar Staggered (10).

Table 4. Eigenstates energies for H_3^{++} with $R=0.25$ au, $\theta = \pi$ (the linear geometry) and $\delta t = 0.02au$

EL	Cartesian	Pol-NS($N_t = 4$)	Pol-NS($N_t = 8$)	Pol-S($N_t = 4$)	Pol-S($N_t = 8$)
0	-3.2240457503	-3.2312695029	-3.2312902965	-5.0314760467	-3.2335929632
0	-1.4854989631	-1.4865392462	-1.4865519299	-5.0082106289	-1.4861910636
0	-1.4335007636	-1.4344772184	-1.4344880996	-4.9658053911	-1.4340675482
0	-0.9955385311	-0.9980260407	-0.9980383193	-4.9451699370	-1.3561523102
0	-0.6773706027	-0.9161634664	-0.6774912223	-4.9344122172	-1.2206168110
0	-0.6774407526	-0.8108349361	-0.6773767675	-4.8974305969	-1.1755428580
0	-0.6051399981	-0.6838092821	-0.6055742493	-4.8946511974	-1.0337007211
0	-0.5932459624	-0.6774293616	-0.5936584359	-4.8509200418	-1.0310335933
0	-0.4647300143	-0.6774341861	-0.4656903132	-4.8510265080	-0.9987031681

EL – Energy Level, Pol-NS – Polar Non-Staggered (9), Pol-S – Polar Staggered (10).

sion $\alpha = E_0/m\omega^2$ of an electron of mass m in a field $E(t) = E_0 \cos(\omega t)$ of maximum amplitude E_0 and frequency ω . At $I = 5 \times 10^{14} W/cm^2$, $E_0 = 0.12au$, and for $\omega = 0.057au$ (800nm), $\alpha = 37au$. We compare the MHOHG results of H_2^+/H_3^{++} for the different methods. As we pointed out in the preceding section, we need to use $N_t > 1$ to calculate eigenstates for many cases. In figure 2, we confirm that the propagation in real time can always use $N_t = 1$. We compare the MHOHG calculated with the FFT-Cartesian method, NonStaggered-Polar ($N_t=1$) and Staggered-Polar ($N_t=1$) methods for : a) H_3^{++} ($R=2.0au$, Linear) in figure 3, b) H_3^{++} ($R=4.0au$, Equilateral) in figure 4; c) H_2^+ ($R=2.0au$) in figure 5. For H_3^{++} linear, the NonStaggered-Polar method gets identical results as the FFT-Cartesian method, but the Staggered-Polar method gives worse results because the discrete wavefunction does not have a good approximation at the origin which coincides with proton positions. For the H_3^{++} equilateral case, compared to the results of FFT-Cartesian method, the NonStaggered-Polar method gives better results again than the Staggered-Polar method (9). For the case of H_2^+ , all three methods produce almost identical results (fig 5).

Our last numerical experiment is the simulation of H_2 in 2D in the laser field: $I = 5 \times 10^{14} W/cm^2$, wave length $\lambda = 800nm$, 2 cycles linear turn on + 2 cycles constant + 2 cycles linear turn off. In figure [6] we show the MHOHG calculated with the NonStaggered Polar method [9] with $N_t = 4$ and $N_t = 1$. In all cases, Figs [2][6], we obtain a well defined plateau with a cut-off defined by the semiclassical recollision model around a maximum harmonic N_m with energy $N_m \hbar \omega = I_p + 3.17U_p$, where I_p is the ionization potential and $U_p = E_0/4\omega^2$, the ponderomotive energy in a.u. [13].

References

- [1] Brabec, T., Krausz, F.: Intense Few-Cycles Laser Fields. *Rev. Mod. Phys.* 72, 545 (2000)
- [2] Bandrauk, A.D. (ed.): *Molecules in Laser Fields*. M. Dekker Pub., New York (1994)
- [3] Chelkowski, S., Zuo, T., Bandrauk, A.D.: Ionization Rates of H_2^+ in an Intense Laser Field by Numerical Integration of the Time-dependent Schroedinger Equation. *Phys. Rev. A*46, R5342 (1992)
- [4] Feit, M.D., Fleck, J.A.: Solution of the Schroedinger Equation by a Spectral Method II: Vibrational Energy Levels of Triatomic Molecules. *J. Chem. Phys.* 78, 301 (1983)
- [5] Zwillinger, D. (ed.): *CRC Standard Mathematical Tables and Formulae*, pp. 711–714. CRC Press, N.Y (1996)
- [6] Zuo, T., Bandrauk, A.D., Ivanov, M., Corkum, P.B.: Control of High-order Harmonic Generation in Strong Laser Fields. *Phys. Rev. A*51, 3991 (1995)
- [7] Bandrauk, A.D., Lu, H.Z.: Singularity Free Methods for the Time-Dependent Schroedinger Equation. In: Bandrauk, A.D., Le Bris, M.C. (eds.) *High Dimensional PDE's*. CRM Proceedings, vol. 41, pp. 1–14. AMS, Providence (2007)
- [8] Dion, C.: Simulation Numérique de l'Isomérisation de HCN en Champ Laser Intense (Mémoire de maîtrise en sciences, Univ. de Sherbrooke) (1995)
- [9] Kono, H., Kita, A., Ohtsuki, Y., Fujimura, Y.: An Efficient Quantum Mechanical Method for the Electronic Dynamics of the Three-dimensional Hydrogen Atom Interacting with a Linearly Polarized Strong Laser Pulse. *J. Comp. Phys.* 130, 148 (1997)
- [10] Harumiya, K., Kono, H., Fujimura, Y., Kawata, I., Bandrauk, A.D.: Intense laser-field ionization of H_2 enhanced by two-electron dynamics. *Phys. Rev. A* 66, 043403 (2002)
- [11] Bandrauk, A.D., Lu, H.Z.: *Handbbok of Numerical Analysis – Computational Chemistry*. Ciarlet, P.G., LeBris, C. (eds.), vol. X, pp. 803–830. Elsevier Science, B.V., Amsterdam (2003)
- [12] Bandrauk, A.D., Shen, H.: Exponential Split Operator Methods for Solving Coupled Time-dependent Schrodinger Equations. *J. Chem. Phys.* 99, 1185 (1993)
- [13] Bandrauk, A.D., Barmaki, S., Lagmago Kamta, G.: Molecular Harmonic Generation. In: Yamanouchi, K. (ed.) *Progress in Ultrafast Intense Laser Science*, ch. 9, vol. III. Springer, NY (2008); *J. Chem. Phys.* 99, 1185 (1993)

A Parallel Algorithm for Computing the Spectrum of CH_5^+

Xiao-Gang Wang and Tucker Carrington Jr.

Chemistry Department, Queen's University, Kingston, Ontario K7L 3N6, Canada
Fax: 613-533-6669

Xiaogang.Wang@umontreal.ca, Tucker.Carrington@chem.queensu.ca

Abstract. We present a parallelized contracted basis-iterative calculation of vibrational energy levels of CH_5^+ (a 12D calculation). We use Radau polyspherical coordinates and basis functions that are products of eigenfunctions of bend and stretch Hamiltonians. The basis functions have amplitude in all of the 120 equivalent minima. Many low-lying levels are well converged. A new parallelization scheme is presented.

1 Introduction

Protonated methane, CH_5^+ , has fascinated scientists for years [12,34] because it is very different from most molecules. It is well-bound, but has 120 equivalent minima separated by small barriers. The wavefunctions have significant amplitude in all of the wells. The minimum energy shape has C_s symmetry and an H_2 group on top of a CH_3 unit. Two classes of low-energy pathways connect the minima. [5] There are internal rotation pathways with Cs saddle points that roughly correspond to rotating an H_2 group. There are “flip” [5], pathways (with C_{2v} saddle points) that exchange a proton between a CH_3 tripod and a H_2 group. The internal rotation barrier height is about 40 cm^{-1} and the flip barrier is near 300 cm^{-1} . [6] To compute a spectrum of CH_5^+ it is imperative that one use a method capable of dealing with the large amplitude motion of the protons.

Knowing the potential energy surface it is in principle possible to solve the (ro)vibrational Schroedinger equation to compute the (ro)vibrational spectrum. To do this one must choose coordinates, derive a kinetic energy operator (KEO), select basis functions with which to represent wavefunctions, and solve a matrix eigenvalue problem. It is difficult to compute vibrational spectra of molecules because of the dimensionality of the associated Schroedinger equation. For CH_5^+ the equation is 12-dimensional. To compute numerically exact solutions for CH_5^+ it is essential to choose basis functions that have significant amplitude in all the 120 wells among which the protons move. [4] This, in conjunction with the fact that there are 6 atoms, means that the required basis is huge. It is impossible to use a basis of products of functions of a single coordinate. [7] If only 10 basis functions were sufficient for each coordinate, a basis of 10^6 functions would be necessary to compute the $J = 0$ levels of a four-atom molecule, 10^9 functions for a five-atom molecule, 10^{12} functions for a six-atom molecule etc. This is a manifestation of the “curse of dimensionality”.

2 Contracted Basis Set Method

In this report we use a contracted basis set, the Lanczos algorithm and a potential developed by Braams and Bowman and co-workers. [6] A simple product basis [7] is far too large. Even if only 10 functions per coordinate were sufficient one would need 8 TB of memory for a single vector in 12D. Some sort of contraction scheme is therefore essential. In several papers we have shown that it is possible to efficiently do matrix-vector products using a basis of products of bend and stretch functions. [8,9] If polyspherical coordinates are used it is possible to write the full Hamiltonian

$$H = H^{(b)} + H^{(s)} + \Delta T + \Delta V \quad (1)$$

where

$$\Delta V(r, \theta) = V(r, \theta) - V_{bref}(\theta) - V_{sref}(r) \quad (2)$$

and

$$\Delta T = \sum_i \Delta B_i(r) T_b^{(i)}(\theta) . \quad (3)$$

with

$$\Delta B_i(r) = B_i(r) - B_i(r_e) . \quad (4)$$

$H^{(b)}$ is the bend Hamiltonian obtained from the full Hamiltonian by discarding the stretch kinetic energy term, choosing an appropriate reference bend potential $V_{bref}(\theta)$, and evaluating the B in the bend kinetic energy operator (KEO), Eq. (5), at a reference value for the stretch coordinates (denoted r_e). The bend KEO is,

$$T_{ben} = \sum_i B_i(r) T_b^{(i)}(\theta) , \quad (5)$$

where $T_b^{(i)}(\theta)$ are differential operators. $H^{(s)}$ is the stretch Hamiltonian obtained from the full Hamiltonian by discarding the bend kinetic energy term and choosing an appropriate reference stretch potential $V_{sref}(r)$. The product contracted (PC) basis functions we use are products of eigenfunctions of $H^{(b)}$ and $H^{(s)}$. For both the stretch and the bend we retain eigenfunctions with eigenvalues below a threshold.

3 Parallelization of the Bend Problem

Even solving the Schroedinger equation for $H^{(b)}$ is an extremely difficult problem. To define the reference bend potential we find the Radau lengths that, for each set of angles minimize the potential. We choose the Lanczos algorithm [10,11] and evaluate matrix-vector products in a parity adapted basis whose functions are combinations of primitive bend basis functions that are products of spherical harmonics and an associated Legendre function. We have used primitive bend bases with $l_{max} = m_{max}$ (maximum value of l and m) = 17 and 18. [12] Potential matrix-vector products are computed using ideas similar to those

in Ref. [13]. A potential ceiling was imposed at $V_b^{ceil} = 12000 \text{ cm}^{-1}$ to reduce the spectral range of the bend Hamiltonian matrix. Bend eigenfunctions are computed and values at quadrature points at which the V_{bref} value is smaller than $V_b^{ceil} \text{ cm}^{-1}$ are stored on disk (compaction method). [9][14] This reduces the space required to store the computed bend functions by about a factor of 8.

The molecular symmetry group for the bend problem G_{240} is a direct product of the inversion group and the permutation group of five particles S_5 . [15] The coordinate symmetry group we use includes only operations that permute four protons i.e., $S_4 \otimes \{E, E^*\}$. We compute bend levels and eigenfunctions for 10 symmetry species $A_1^\pm, A_2^\pm, E_a^\pm, F_{1x}^\pm, F_{2z}^\pm$ in parallel, using projection operators to purify the Lanczos vectors. [16].

For each symmetry species the matrix-vector product is parallelized over a quadrature index and vectors labelled by different values of this quadrature index are computed on different processors. This is implemented by using OPENMP and denoted Method I. This permits efficient parallelization and also reduces the size of intermediate vectors. The bend matrix-vector product we must compute has the same form as the methane bend matrix-vector product in Eq 15 of Ref. [13]:

$$x'_{l'_1, l'_2, m'_2, l'_3, m'_3, l'_4, m'_4} = \sum_{\alpha_1} T_{l'_1 \alpha_1}^{(m'_1)} y_{l'_2, m'_2, l'_3, m'_3, l'_4, m'_4}^{(\alpha_1)} \quad (6)$$

where

$$\begin{aligned} y_{l'_2, m'_2, l'_3, m'_3, l'_4, m'_4}^{(\alpha_1)} &= \sum_{\alpha_2} T_{l'_2 \alpha_2}^{(m'_2)} \sum_{\alpha_3} T_{l'_3 \alpha_3}^{(m'_3)} \sum_{\alpha_4} T_{l'_4 \alpha_4}^{(m'_4)} \sum_{m_2, m_3, m_4} I_{m'_2 m'_3 m'_4, m_2 m_3 m_4}^{\alpha_1 \alpha_2 \alpha_3 \alpha_4} \\ &\times \sum_{l_1} T_{l_1 \alpha_1}^{(m_1)} \sum_{l_2} T_{l_2 \alpha_2}^{(m_2)} \sum_{l_3} T_{l_3 \alpha_3}^{(m_3)} \sum_{l_4} T_{l_4 \alpha_4}^{(m_4)} x_{l_1, l_2, m_2, l_3, m_3, l_4, m_4} \end{aligned} \quad (7)$$

To compute the \mathbf{I} matrix we must do 3D fast Fourier and inverse fast Fourier transforms. We use OPENMP to parallelize over α_1 . Two finite basis representation (FBR) vectors (\mathbf{x} and \mathbf{x}' in the above equations) and the potential vector are shared across the threads, all other operations are local within each thread. Computing \mathbf{y} vectors for different α_1 on different processors not only permits efficient parallelization, it also reduces the size of mixed FBR-discrete variable representation (DVR) intermediate vectors by a factor equal to the number of values of α_1 . This is critical because the mixed FBR-discrete variable representation (DVR) vector is about 2^4 times larger than the initial FBR vector. By parallelizing over 10 irreducible representations (irrep), and for each irrep over the outer quadrature index using 16 OPENMP threads, the bend problems are efficiently solved with a total of 160 threads. Overall, the parallelization speeds up the calculation by about a factor of 200.

This new parallelization scheme is more efficient than alternatives. The most obvious way to parallelize is to divide the bend basis functions into groups and to compute pieces of the matrix-vector product for different groups on different processors. This would be inefficient with our basis because our matrices are dense. No matter how the groups are defined a lot of communication between

processors would be necessary. Another alternative is to parallelize each of the sums in Eq. (6) and Eq. (7), or in a slightly better version, combine the four right-most sums under a common loop over m_2, m_3, m_4 and parallelize it, combine the four left-most sums under a common loop over m'_2, m'_3, m'_4 and parallelize it. We denote this Method II. This is not a good strategy because it requires a lot of communication between processors. Imagine that the vector x on the right side of Eq. (7) is stored in a matrix with columns labelled by m_2, m_3, m_4 values and rows labelled by l_1, l_2, l_3, l_4 values. One or more columns are stored in the memory of a single processor. Applying the T matrices transforms data in a single column and hence on one processor. One obtains a data structure whose columns are still labelled by m_2, m_3, m_4 values but whose rows are now labelled by $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ values: elements with different m_2, m_3, m_4 values are stored on different processors. To apply the FFT, i.e., sum over m_2, m_3, m_4 , the computer must access elements on different processors and this implies a lot of communication. The same problem hampers the parallelization of the plane wave electronic structure calculations. [17][18]

Table 1. Times (in seconds) for computing a bend potential matrix-vector product for CH_5^+ . The bend basis size is 215 million, determined by $l_{max} = m_{max} = 18$. The timing is obtained on a 4-way quad-core AMD Opteron (Barcelona, 2 GHz) computer with 4 GB memory per core. The number in parentheses is the ratio of time to the time with a single core.

# of cores	Method I	Method II
1	876 (1.0)	843 (1.0)
2	454 (1.9)	456 (1.9)
4	238 (3.7)	240 (3.5)
8	137 (6.4)	133 (6.3)
15	79 (11.1)	111 (7.6)

Table II and Figure II show how the time to do a potential matrix-vector product scales as a function of the number of processor cores on a 4-way quad-core AMD Opteron (Barcelona, 2 GHz) computer with 4 GB memory per core. An additional core is assigned to do the kinetic matrix-vector product at the same time as the potential matrix-vector product is computed. Doing the kinetic matrix-vector product takes much less time than doing the potential matrix-vector product. As seen in Figure II, the new method (method I) and the parallelization of the sums (method II) perform similarly when the number of cores is smaller than 8, and but the new method is about a factor of two better than method II when the number of cores is 15. This shows that when the number of core increases, the overhead of accessing data across processors in method II becomes significant. Because the calculations are so time consuming even a small speed-up has important consequences. To compute even the lowest bend levels and their associated eigenfunctions requires about 1000 matrix-vector products. To do this for all irreps takes about a month of CPU time.

Table 2. Vibrational energy levels in cm^{-1} of CH_5^+ .

Sym (degen.)	bend	full-d
$A_1^+ (1)$	3826.2	10926.5
$H_1^+ (5)$	22.0	21.7
$G_1^+ (4)$	45.6	39.3
$H_2^+ (5)$	57.1	52.3
$I^+ (6)$	101.8	89.4
$H_1^+ (5)$	116.7	106.5
$G_1^+ (4)$	158.4	153.0
$G_2^- (4)$	10.4	10.4
$H_2^- (5)$	42.1	39.8
$I^- (6)$	54.1	47.3
$G_2^- (4)$	99.5	85.6
$H_1^- (5)$	106.8	96.2
$H_2^- (5)$	142.2	137.1

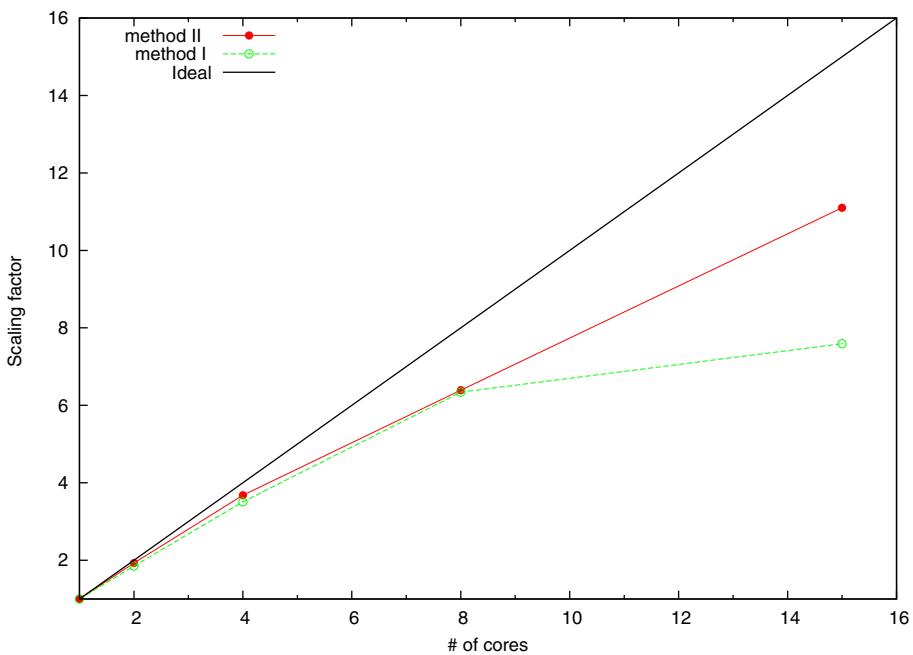


Fig. 1. Time scaling factor for computing a bend potential matrix-vector product. The bend basis size is 215 million, determined by $l_{max} = m_{max} = 18$. The timing is obtained on a 4-way quad-core AMD Opteron (Barcelona, 2 GHz) computer with 4 GB memory per core. The time of the two methods for 1 core is similar.

4 Solving the Stretch Problem and Using the PC Basis

It is relatively easy, using the symmetry adapted Lanczos algorithm [16], to compute stretch eigenfunctions that transform like irreps of $S_2 = [E, (23)]$, where H₂ and H₃ are two of the H atoms not used to specify the body-fixed frame used to define the polyspherical coordinates. The stretch reference potential is defined by fixing the bend angles to those of the C_{2v} saddle point. Products of 5 1-d potential optimised discrete variable representation (PODVR) [19] functions are used as primitive stretch basis functions. The number of primitive stretch basis functions is reduced to 2477 by removing PODVR functions associated with points at which the potential is larger than 20000 cm⁻¹.

We use 12-d PC basis functions that transform like irreps of $S_2 \otimes \{E, E^*\}: A^+, B^+, A^-$ or B^- . If the levels are well converged G_{240} labels can be determined using the correlation between S_5 and S_2 . We retain 630 stretch functions with energies less than 20000 cm⁻¹ and 2883 bend functions with energies less than 3400 cm⁻¹. A costly part of the full-d calculation is the computation of the \mathbf{F} matrix. We use symmetry to reduce the number of \mathbf{F} elements we compute and the cost of using \mathbf{F} to evaluate matrix-vector products. \mathbf{F} elements are computed from compacted bend functions stored on disk. For each bend function only values at quadrature points at which the V_{bref} value is smaller than $V_b^{ceil} = 12000$ cm⁻¹ are stored. [9] This reduces significantly the required storage space. MPI is used to parallelize over small groups of bend functions. Each process reads two groups of bend functions from the disk into memory. The reduction in computation time scales linearly because there is no communication between processes. The same idea of dividing bend functions into groups and using MPI is used to compute ΔT matrix elements.

5 Results

Low-lying bend and full-d levels are reported in table 2. All the excited states are reported as differences between the zero point energy (zpe) and the excited state energy. The bend levels are computed from a minimized bend potential with a basis size of 147 million defined by $l_{max} = m_{max} = 17$. The bend levels are converged to about 0.5 cm⁻¹. The full-d levels are not as well converged. The full-d zpe converges slowly. The best converged zpe, 10926.5 cm⁻¹, is about 10 cm⁻¹ above the zpe, 10917.3(3) cm⁻¹, obtained from a diffusion Monte Carlo calculation [4].

The bend and full-d energy levels have similar structure, indicating that the five protons move on a sphere with carbon atom at the centre. The energy level structure is however very different from previous studies using more approximate methods. A great deal remains to be done to fully understand the computed energy levels [4].

6 Conclusion

We have implemented a new strategy for evaluating the bend matrix-vector products required to compute the vibrational spectrum of molecule with more than four atoms. It is more efficient than parallelizing each of sums in Eq. (7). The new idea involves parallelizing over one quadrature index, storing vectors labelled by the quadrature index locally, and then combining them to obtain the result of the matrix-vector product. Using this idea we have computed many vibrational energy levels of CH₅⁺. With a total of 16 cores, the computation is accelerated by a factor of 11. In the future we intend to use similar ideas to compute a rovibrational spectrum of CH₅⁺. This should help to unravel the observed spectra.

Acknowledgments

We are grateful to Joel Bowman for providing his potential energy surface and we thank Anne McCoy for discussions. Most of the calculations were done on a computer of the Réseau Québécois de Calcul de Haute Performance (RQCHP). We thank Jacques Richer and Michel Béland of RQCHP for discussions on parallelization. This work has been supported by the Natural Sciences and Engineering Research Council of Canada and the Canada Research Chairs programme.

References

1. Marx, D., Parrinello, M.: Structural quantum effects and three-center two-electron bonding in CH₅⁺. *Nature (London)* 375, 216 (1995)
2. White, E.T., Tang, J., Oka, T.: CH₅⁺: The Infrared Spectrum Observed. *Science* 284, 135 (1999)
3. Huang, X., McCoy, A.B., Bowman, J.M., Johnson, L.M., Savage, C., Dong, F., Nesbitt, D.J.: Quantum Deconstruction of the Infrared Spectrum of CH₅⁺. *Science* 311, 60 (2006)
4. Wang, X.-G., Carrington Jr., T.: Vibrational energy levels of CH₅⁺. *J. Chem. Phys.* 129, 234102 (2008)
5. East, A.L.L., Kolbuszewski, M., Bunker, P.R.: Ab Initio Calculation of the Rotational Spectrum of CH₅⁺ and CD₅⁺. *J. Phys. Chem. A* 101, 6746 (1997)
6. Jin, Z., Braams, B.J., Bowman, J.M.: An ab Initio Based Global Potential Energy Surface Describing CH₅⁺ → CH₃⁺ + H₂. *J. Phys. Chem. A* 110, 1569 (2006)
7. Bramley, M.J., Carrington Jr., T.: A general discrete variable method to calculate vibrational energy levels of three- and four-atom molecules. *J. Chem. Phys.* 99, 8519 (1993)
8. Wang, X.-G., Carrington Jr., T.: New ideas for using contracted basis functions with a Lanczos eigensolver for computing vibrational spectra of molecules with four or more atoms. *J. Chem. Phys.* 117, 6923–6934 (2002)
9. Wang, X.-G., Carrington Jr., T.: A contracted basis-Lanczos calculation of the vibrational levels of methane: solving the Schrödinger equation in nine dimensions. *J. Chem. Phys.* 119, 101–117 (2003)

10. Paige, C.C.: Computational variants of the Lanczos method for the eigenvalue problem. *J. Inst. Math. Appl.* 10, 373–381 (1972)
11. Cullum, J.K., Willoughby, R.A.: Lanczos algorithms for large symmetric eigenvalue computations. *Theory*, vol. 1. Birkhauser, Boston (1985)
12. The size of the even basis with $l_{max} = m_{max} = 18$ is about 215 million. For this basis, 20 quadrature points are used for the θ coordinates and 40 for the ϕ coordinates
13. Wang, X.-G., Carrington Jr., T.: A finite basis representation Lanczos calculation of the bend energy levels of methane. *J. Chem. Phys.* 118, 6946 (2003)
14. Bramley, M.J., Carrington Jr., T.: Calculation of triatomic vibrational eigenstates: Product or contracted basis sets, Lanczos or conventional eigensolvers? What is the most efficient combination? *J. Chem. Phys.* 101, 8494 (1994)
15. Bunker, P.R., Jensen, P.: Molecular Symmetry and Spectroscopy. NRC Research Press, Ottawa (1998)
16. Wang, X.-G., Carrington Jr., T.: A symmetry-adapted Lanczos method for calculating energy levels with different symmetries from a single set of iterations. *J. Chem. Phys.* 114, 1473 (2001)
17. Clarke, L.J., Štich, I., Payne, M.C.: Large-scale ab initio total energy calculations on parallel computers. *Comp. Phys. Comm.* 72, 14 (1992)
18. Haynes, P., Côté, M.: Parallel fast Fourier transforms for electronic structure calculations. *Comp. Phys. Comm.* 130, 130 (2000)
19. Wei, H., Carrington Jr., T.: The discrete variable representation for a triatomic Hamiltonian in bond length-bond angle coordinates. *J. Chem. Phys.* 97, 3029 (1992)

Modeling the Interactions between Poly(N-Vinylpyrrolidone) and Gas Hydrates: Factors Involved in Suppressing and Accelerating Hydrate Growth

Brent Wathen¹, Peter Kwan², Zongchao Jia¹, and Virginia K. Walker²

¹ Department of Biochemistry, Queen's University
Kingston, Ontario, Canada K7L 3N6

² Department of Biology, Queen's University
Kingston, Ontario, Canada K7L 3N6

Abstract. Gas hydrates represent both a bane and a potential boon to the oil and gas industry, and considerable research into hydrate formation has been undertaken. We have recently developed a multi-threaded version of a Monte Carlo crystal growth algorithm and applied it to simulate the growth of large structure II gas hydrates. This algorithm allows for the introduction of non-crystal molecules during simulations to study their effects on crystal growth rates and morphologies. Here, we report on our initial simulations of hydrate growth in the presence of poly(*N*-vinylpyrrolidone) (PVP). We have surveyed the PVP-hydrate interaction space by performing numerous simulations, each with a unique PVP-hydrate docking orientation. This survey produced a broad range of hydrate growth rates, ranging from almost complete suppression through to significant growth enhancement. The effect that PVP has on gas incorporation at crystal surfaces appears to be critical for differentiating between crystal growth inhibition and enhancement.

Keywords: Gas hydrates, PVP, Monte Carlo simulations, parallel computing.

1 Introduction

Gas hydrates are ice-like crystalline structures composed of water cages surrounding trapped gas molecules. Gas hydrates are troublesome for the petroleum industry since they form when mixtures of gas and water molecules are held at high pressure and low temperature, typical of conditions at oil well heads and in oil and gas pipelines [1]. Hydrate formation in these locations can impede the flow of oil and gas, increasing the cost of flow assurance [1-3] and adding significant hazard to the process of oil recovery [2]. Recently, gas hydrates have been viewed more favourably by others for their gas-trapping abilities. One application, which may have significant impact in the fight against global warming, is the use of hydrates for CO₂ sequestration [4, 5]. Another possible use of hydrates is for the storage and transport of hydrogen gas, a use which could play a substantial role in a future move towards a hydrogen-fueled economy [6]. Moreover, as our supply of readily available petroleum resources continues to dwindle, the huge amounts of methane gas hydrates in the permafrost and under the sea at the

continental margins have the potential to be a massive source of energy if successful extraction techniques can be developed [2]. Collectively, these applications and others are currently fueling interest in the process of gas hydrate formation.

Three distinct forms of hydrates are known to occur naturally, including the cubic structures I and II (sI and sII), and the much rarer hexagonal structure sH [7]. All three are built from a combination of basic 5^{12} "small" cage building blocks (12-faced pentagonal dodecahedron structures) with other water cage structures. In the case of sII, 16 of the 5^{12} small cages combine with 8 $5^{12}6^4$ "large" cages in a periodic manner to produce a crystalline structure that has a unit cell with 136 water molecules and 24 small and large gas-trapping cages (Figure 1). While gas hydrates are non-stoichiometric compounds, the gas occupancy rate generally approaches 100% [8]. It is not possible at present to determine *a priori* which of these three structures will form for a given mixture of gas and water [9], particularly if multiple gases are present. However, in general, smaller gas molecules tend to promote sI formation, while larger ones tend to promote sII formation [10].

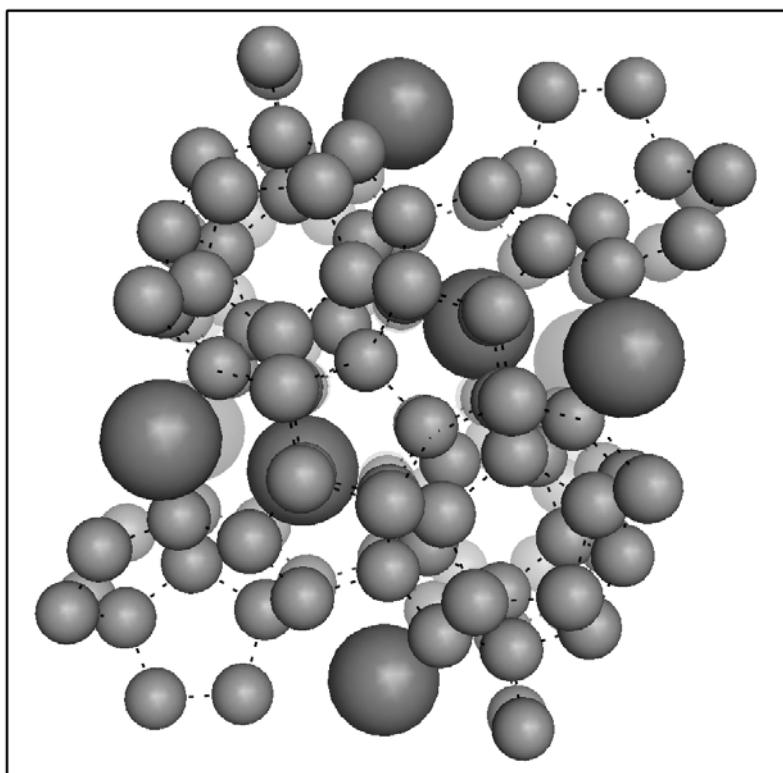


Fig. 1. The unit cell of the Type II gas hydrate. There are 136 water molecules (smaller spheres) in the Type II unit cell, forming small and large cages. Eight large cages trap gas molecules (larger spheres), two of which are shown fully enmeshed trapped in this figure (central two large spheres). Dashed black lines show hydrogen bonding between neighbouring water molecules.

Because of the opposing desires both to suppress (for flow assurance in the petroleum industry) and enhance (for applications such as CO₂ sequestration and H₂ storage/transport) gas hydrate growth, considerable effort has been expended to control hydrate formation. Formation rates are typically low because of the poor gas solubility in aqueous solutions [11]. To increase hydrate formation rates, for example, agitation, seeding, and the use of surfactants have all been studied [11]. In contrast, to decrease hydrate formation rates, two different classes of hydrate inhibitors have been employed. The first class includes thermodynamic inhibitors such as alcohols and glycols that are widely used by industry to minimize gas hydrate growth and enhance petroleum flow [1]. Large quantities of these inhibitors are required, however, to achieve hydrate growth inhibition, making their use costly [2]. A more recently developed second class of inhibitors, low-dosage hydrate inhibitors (LDHIs), reduce costs of flow assurance [12]. These two classes of inhibitors work by different mechanisms: thermodynamic inhibitors alter the physical properties of the medium, effectively decreasing the amount of water available for hydrate formation through hydrogen bond competition [2], while LDHIs appear to disrupt hydrate growth following formation [13], even changing the morphology of clathrates in the laboratory [e.g. 14, 15].

A detailed understanding of the gas hydrate nucleation and growth mechanisms would greatly assist in the development of strategies to manage hydrate formation. Unfortunately, these formation mechanisms remain poorly understood at present, particularly when formation occurs in the presence of a mixture of gases. To address this problem, numerous molecular dynamics (MD) simulations of hydrate formation have been undertaken. One such study has shown that structures resembling sII can be present at early stages of sI formation [16]. Several MD studies have examined hydrate stability [17, 18], while others have investigated the mechanisms of LDHI inhibition [1, 13, 19]. These latter simulations have shown that some LDHIs, such as poly(N-vinylpyrrolidone) (PVP), function by binding to gas hydrates and arresting subsequent growth [13], while others can affect hydrate growth by organizing surrounding water molecules in a way that is inconsistent with hydrate formation [19]. Because of limited computational resources, these simulations are restricted to small numbers of molecules, perhaps hundreds or thousands of water and gas molecules, and one or only a few inhibitor molecules. In particular, they are not suitable for investigation of the inhibitory effects of up to thousands of inhibitor molecules working cooperatively.

Previously, we have investigated crystal growth using a novel Monte Carlo (MC) computational approach that simulates the growth of large, three-dimensional (3D) crystals, rather than isolated crystal surfaces [20, 21]. This approach allows for the inclusion of large, non-crystal molecules with realistic (though fixed) shapes and interaction energetics in simulations. Here, we report on recent enhancements to our algorithm that take advantage of parallel computing environments, and its application to the study of gas hydrate growth in the presence of PVP polymers.

2 Methods

2.1 Whole-Crystal Monte Carlo Simulations

The whole-crystal Monte Carlo (MC) simulation algorithm has been previously described [20, 21]. In brief, the underlying model for this technique is based on the

Kinetic Ising model [22]. At the heart of our approach is the *simulation space* (Figure 2A), a 3D matrix describing the spatial arrangement of crystal locations and the connectivity between these locations. Connected positions are referred to as neighbours. The particular structure of the simulation space is chosen to reflect the molecular structure of a crystal under investigation.

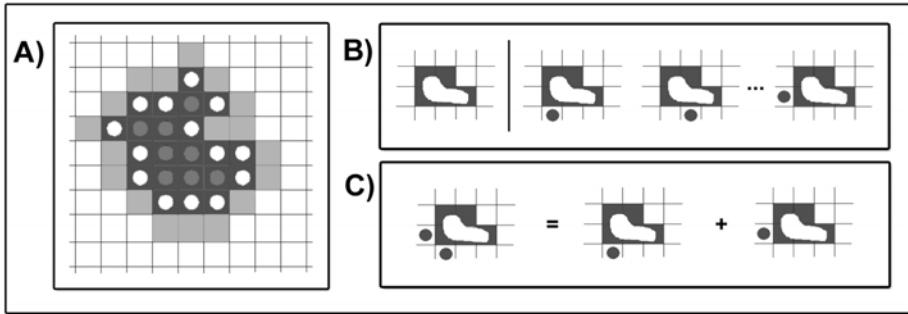


Fig. 2. MC Simulations. A) A simplified two-dimensional simulation space, with left-right-up-down neighbouring connectivity. Occupied (dark grey boxes), surface (white spheres), buried (grey spheres) and interface (light grey boxes) positions are shown. B) A non-crystal molecule (NCM) consists of its volume set (left, dark grey squares), and its energetics set (right, depicting the interactions with each of its neighbours (white squares, dark grey spheres), determined individually). C) During simulations, the interaction energy between a bound NCM and a growing crystal is the sum of the interaction energies between the NCM and each of its occupied neighbouring positions.

An initial seed crystal is placed in the center of the simulation space. Subsequently, each step of the simulation will either (i) remove a molecule from the crystal surface (a dissociation event), or (ii) add a molecule to an interface location neighbouring a surface molecule (an association event). Probability is used to determine what type of event takes place in each step. Association events happen with probability P_{assoc} :

$$P_{assoc} = Loc_{interface} * P_{on} \quad (1)$$

where $Loc_{interface}$ is the total number of interface locations, and P_{on} is a fixed probability of a molecule joining the crystal. Dissociation events happen with probability P_{dissoc} :

$$P_{dissoc} = P_{off}(number\ of\ occupied\ neighbour) \quad (2)$$

where P_{off} , the probability of dissociation at a particular location, is a function of the number of occupied neighbouring positions. $P_{off}(0)$ is set to 1.0 to ensure that any molecule that becomes "stranded" is removed from the simulation, while $P_{off}(\text{full occupancy})$ is set to 0.0 to ensure that "buried" molecules (ie. non-surface molecules) do not dissociate. P_{off} values for intermediary neighbouring occupancies are given by:

$$P_{off}(i) = \exp[-i\varphi/k_B T] \quad (3)$$

where i is the number of neighbouring locations that contain crystal molecules, φ is the bond energy, k_B is Boltzmann's constant and T is temperature. In this manner, the underlying connectivity within a crystal structure is incorporated into dissociation probabilities, thereby enabling it to affect morphology and other growth characteristics during simulations.

The basic model has been extended so that non-crystal molecules (NCMs) can be incorporated into simulation spaces during simulations. NCMs differ from crystal molecules in two critical aspects. First, they are much larger than crystal molecules, typically occupying hundreds of times more volume than crystal molecules. Secondly, their energetics can be non-uniform, with varying attractive and repulsive forces between themselves and neighbouring crystal molecules across their surfaces. NCMs are first created from a docking between a molecule of interest (ie. a nucleator or an inhibitor) and the crystal, from which a fixed volume set is determined (Figure 2B). Next, an energetics set describing the interaction energies between a docked NCM and all neighbouring simulation space locations is determined (Figure 2C). There is a fixed association probability P_{assoc} (reflecting inhibitor concentration, diffusion through the liquid medium and the chance of a correctly oriented collision with the crystal) for NCMs, while the dissociation probability P_{dissoc} for a particular bound NCM is determined at run-time based on which specific neighbouring locations are occupied by crystal molecules. It must be emphasized that NCMs are static in simulations: their shape and binding orientation are predetermined when they are created, and remain fixed during simulations.

2.2 Model Implementation Changes Required for Gas Hydrate Simulations

In order to adapt the model for studying gas hydrate growth, a number of changes have been made to the original methodology. Generally, these are the result of (*i*) differences in the underlying crystal structures of ice and gas hydrate, and (*ii*) the lack of a detailed PVP-hydrate docking model.

Hydrate sII has a structure that is inherently more complex than that of ice *Ih* due to the presence of the trapped gas molecules. Because of the fundamental role of the gas in hydrate formation, the algorithm has been modified to allow for multiple types of crystal molecules, each given an independent association parameter. This model enhancement allows the effects of gas concentration on crystal growth rates to be investigated, and provides a simple test of model correctness: gas hydrate crystals should melt in simulations without sufficient gas concentrations (corresponding to gas pressures *in situ*). More generally, the larger, more complex sII unit cell required a redesign of the data structures used to describe the crystal, and of those used to manage the simulation space.

A more significant obstacle to studying hydrate formation in the presence of NCMs is the lack of published experimental evidence supporting any specific docking model between a gas hydrate crystal and an inhibitor or nucleator. Without such a model, meaningfully oriented NCMs molecules cannot be created for inclusion in MC simulations. To circumvent this problem, we have performed a large-scale screening process of possible docking models between NCMs based on the PVP inhibitor molecule, and type sII gas hydrates in order to identify factors that affect hydrate growth. This screening process is extremely computationally demanding. To

ensure the feasibility of this approach, we have redesigned the algorithm to take advantage of parallel processing environments. However, the overall methodology of the MC simulation model does not lend itself well to temporal parallelization, primarily because each simulation step is dependent on the outcome of all prior simulations steps, necessitating serial treatment. Rather than temporal, our approach to distributing the workload across multiple processors is *spatial*: we partition the simulation space into symmetrical regions and assign one processor to manage the simulation in each region. Two complications arise from this approach. First, because association and dissociation events can affect the status of neighbouring locations, there are complications at regional boundaries. To resolve this issue, we implemented a message-passing scheme between processors to ensure that changes to the simulation space are properly coordinated across adjacent regions. The second complication that arises from spatial partitioning of simulation space pertains to NCMs: such molecules may residue in multiple regions because of their extended volume. The association or dissociation of a particular NCM would thus be a costly event in a parallel environment, involving the coordinated control over a number of processors. Costliness, coupled with the relative rarity of inhibitor events, suggested a compromise solution: NCM associations and dissociations can be handled most easily by temporarily interrupting parallel processing. Thus, the basic algorithm for our model was adapted to follow a repetitive sequence of two primary steps: first, a random number of crystal association/dissociation steps is performed in parallel across all regions of the simulation space (the actual number of steps being based on the relative probabilities of crystal molecule and NCM association/dissociation events); following this, parallel processing is interrupted to allow for a single NCM association/dissociation event.

2.3 Simulation Method

A series of whole crystal MC simulations were performed to investigate the effects of PVP on hydrate sII formation. The unit cell of hydrate sII contains 136 water molecules, arranged so as to form 16 small and 8 large cages (Figure 1). A simulation space structure was adopted that combined the spatial positioning and connectivity of the hydrate sII water molecules, with additional positions for gas molecules located at the center of each of the eight large cages, giving a total of 144 water and gas positions in the simulation space unit cell. Each of the eight central positions within the large cages was connected to the 28 water locations that comprise each large cage. The strength of each of these connections was set to 1/8th of the connection strength between neighbouring water molecules, reflecting the weaker dispersive forces acting between gas and water molecules in hydrate crystals.

All simulations were run at 277K. As a first approximation, we adopted both the hydrogen bond energy value and the water association rate (i.e. the probability that a water molecule will join the crystal) from our previous ice growth studies. In a first set of simulations that lacked inhibitor molecules, the gas association rate (i.e. the probability that a gas molecule will join the crystal) was varied to explore the effect of this parameter on hydrate growth rates and gas occupancy. The range of variance was between 0.625% and 100% of the water association rate.

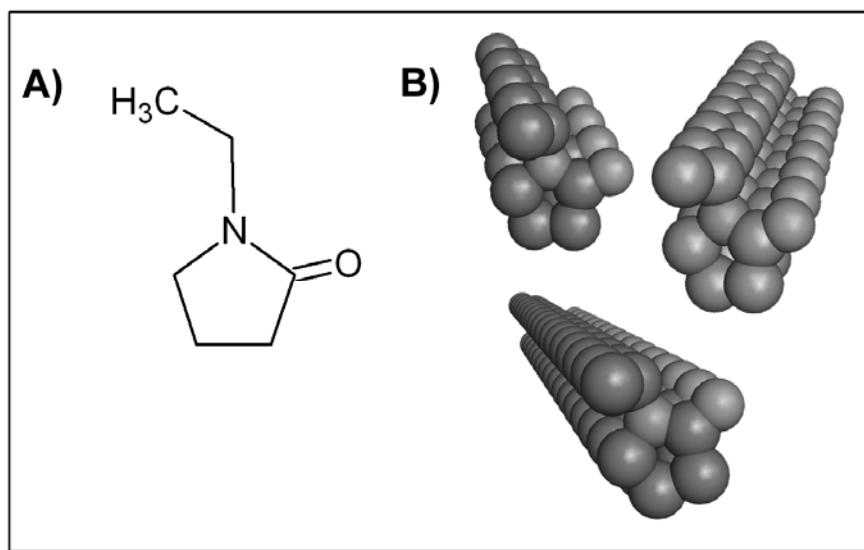


Fig. 3. Poly(N-vinylpyrrolidone) (PVP). A) The basic chemical structure of a PVP monomer. B) Three dimensional view of (clockwise from top left) the 4-monomer, 8-monomer and 16-monomer PVP molecules used in simulations. Extended conformations were chosen that stack the ring structures together and create a hydrophobic side for hydrate binding. This hydrophobic binding face is visible on the left side of the 16-monomer PVP.

PVP polymers (Figure 3) containing 4, 8, and 16 monomers were used to create simulation NCMs to study the effects of polymer length on crystal growth. For initial inhibitor studies, an easily extendible PVP conformation was selected, and a notably flat, hydrophobic surface of this conformation was selected as the docking face for hydrate interactions. Volume sets were created for these NCMs by determining which simulation space locations were carved out by each PVP polymer in a standardized docking orientation. Neighbouring sets were created by discovering which simulation space locations were adjacent to the volume set locations. Neighbours to the hydrophobic binding surface were given attractive interaction energies equal to 1/4th of the connection strength of a hydrogen bond; all other neighbours were given neutral interaction energies, neither attractive nor repulsive.

The PVP 16-mer (PVP16) was used in all subsequent PVP simulations. An initial screening of 512 PVP16 docking orientations was conducted, each obtained by rotating the PVP16 polymer through 16 different angles in one rotation dimension (0° through 337.5°, at intervals of 22.5°), 8 different angles in a second dimension (0° through 157.5°, at intervals of 22.5°), and finally 4 different angles in a third dimension (0° through 135°, at intervals of 45°). Volume and energetics sets were created as described above. Gas hydrate crystals in these initial screening simulations exhibited a broad spectrum of growth rates, from almost complete growth inhibition to substantially increased growth. From these simulations a series of narrower angle ranges were then selected for more targeted screenings. Ranges were chosen that exhibited both enhanced and suppressed growth in the initial screenings. PVP16

NCMs were created at 1° intervals within these ranges in an effort to identify pairs of highly similar NCMs that produced dramatically different effects on growing crystals. Several such pairs were identified, and their effects on gas hydrate growth were investigated.

All simulations began with initial seed crystals containing 100 000 water and gas molecules, and were run for 1 billion simulation steps. All software was written in the 'C' programming language. Simulations were run on the SunFire Cluster (a symmetric multiprocessor environment based on Sun's UltraSPARC processor, running the Solaris Operating Environment) at the High Performance Computing Virtual Laboratory (HPCVL) based in Kingston, Ontario. Parallelization was achieved using OpenMP.

3 Results and Discussion

We have modified our existing MC crystal growth simulation software to study gas hydrate growth in the presence of large, non-crystal molecules (NCMs). Most significantly, the simulation software was parallelized to enhance the investigation of NCM effects on crystal growth in a wide screening of possible docking orientations. Parallelization was achieved by spatially partitioning the simulation space into symmetric regions and by assigning one processor to manage crystal growth simulations in each region. Performance was found to increase well with the use of up to 16 processors (Table 1); beyond 16 processors, little performance improvement was seen. This is not unexpected, since an increase in the number of processors used results in a larger proportion of regional boundary positions and a concomitant increase in the amount of inter-process communication required for data structure management.

Table 1. Simulation run times^a as a function of the number of processors

Number of Processors	Run Time (s)	Efficiency ^b
1	2668	1.00
2	1443	0.92
4	824	0.81
8	510	0.65
16	357	0.47
24	326	0.34
48	327	0.17

^a A Identical simulation conditions (1 million molecule seed crystals, simulation temperature 277K, 500 million simulation steps) were used for all runs.

^b Efficiency is measured by:

$$T_{\text{serial}} / (T_{\text{parallel}} * \text{no. processors}).$$

3.1 Gas Hydrate Simulations without NCMs

The use of accurate association and dissociation rate parameters for all simulation molecules is critical for achieving realistic simulation results. As indicated, the water association and dissociation rate values from our previous studies of ice crystal growth were used. Although the higher temperatures used for gas hydrate growth simulations (277K versus 270-273K for ice growth simulations) has some influence on hydrogen bond energy and water association rate, a large effect is not anticipated. With regards to the gas molecules, an initial series of simulations was performed to determine an appropriate range for the gas association-rate parameter. All of these simulations started with a 100 000 molecule gas hydrate crystal and ran for 3 billion simulation steps, with the gas association rate varying from 0.625% to 100% of the water association rate parameter. As seen in Figure 4, this had a dramatic effect on hydrate growth rates.

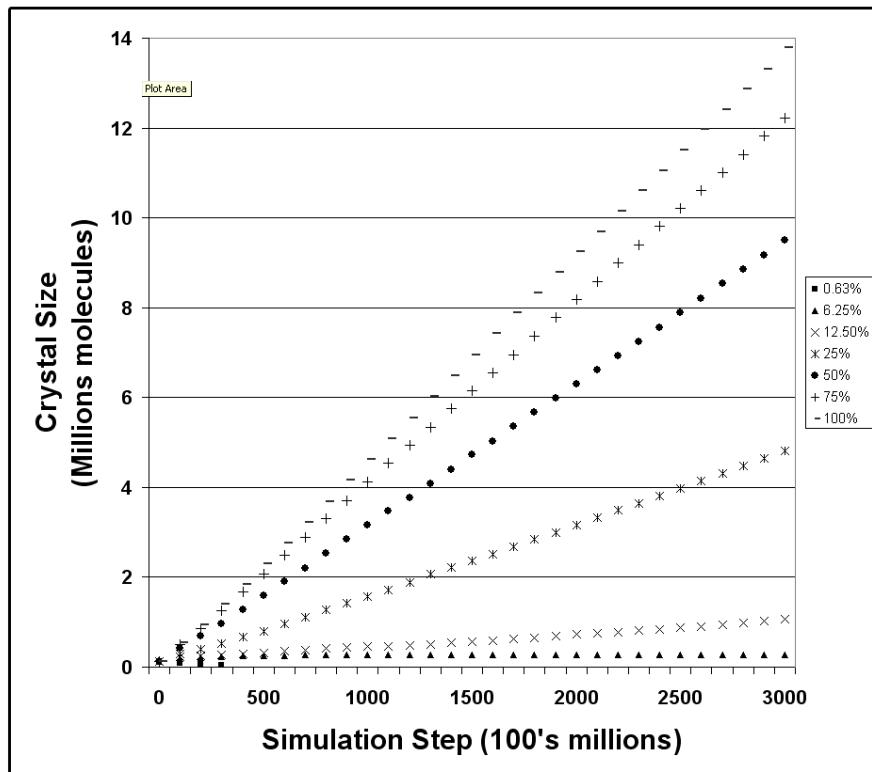


Fig. 4. Gas hydrate growth as a function of the gas association rate (expressed as a percentage of the water association rate)

Hydrate crystals were stable with the gas association rate set to 6.25% of the water association rate; lower values caused the hydrate to melt, while higher values caused it to grow. The water:gas ratio in the resulting crystals was also dependent on the gas

association rate parameter, ranging from 22.0:1 to 17.1:1 (amongst those crystals that did not melt) as the gas association rate was increased. Although experimental results suggest that gas hydrate cage occupancy is near complete (full large cage occupancy yields a water:gas ratio of 17:1), we elected to use a somewhat lower association rate in subsequent simulations to reduce the gas hydrate growth pressure.

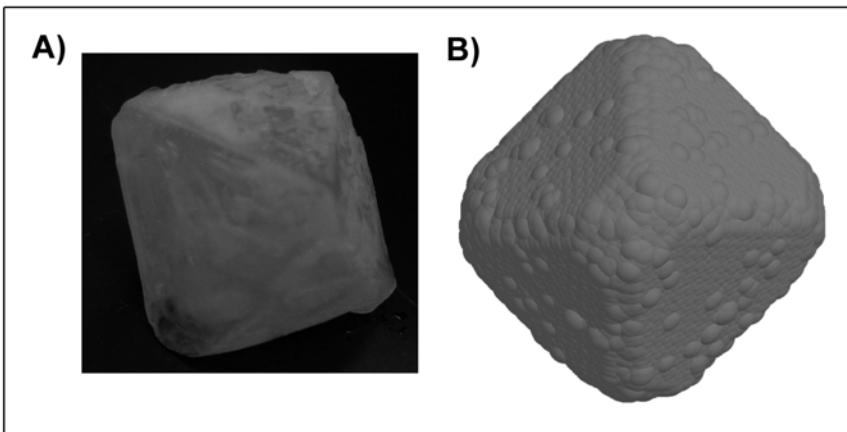


Fig. 5. Single octahedral crystals of gas hydrates, grown A) *in situ* (photo courtesy of Raimond Gordienko), and B) in simulations. In B), water molecules are shown as red spheres, and incompletely trapped gas molecules are shown as gold spheres.

All crystals grown in these initial simulations had similar octahedral morphologies (Figure 5). Unlike in previous simulation work with ice *Ih* crystals where the crystal morphology could be altered by changes to the water rate parameters, we found the morphology of gas hydrate crystals to be highly resistant to change. In these initial explorations of parameter space, only octahedral crystals were observed.

3.2 Gas Hydrate Simulations with NCMs

NCMs based on the known kinetic inhibitor PVP were introduced into simulations to investigate factors that may affect hydrate growth. PVP is a polymer consisting of a hydrocarbon backbone connected to 2-pyrrolidone ring structures (Figure 3). As mentioned, there have been some previous molecular simulation and experimental studies to investigate PVP, however, the nature of the interaction between PVP and hydrate, the PVP polymeric state best suited to hydrate inhibition, and the PVP inhibition mechanism are presently not understood.

We first examined the effect of PVP size on gas hydrate growth by running a series of simulations using NCMs modeled on different PVP polymer lengths. Figure 6 shows the growth of gas hydrates in the absence of NCMs, and in the presence of 4-, 8- and 16-monomer PVP NCMs. The mid-size 8-monomer PVP NCM (PVP8) appeared to achieve complete hydrate growth inhibition, while the smaller 4-monomer PVP NCM (PVP4) showed no appreciable effects on hydrate growth. The 16-monomer PVP NCM

(PVP16) had intermediary effects on hydrate growth, noticeably reducing growth without completely suppressing it. The smaller size of PVP4 results in a smaller surface contacting a hydrate crystal, and we speculate that this leads to weakened overall interactions and higher dissociation rates. Conversely, we suspect that the PVP8 NCM achieves strong growth inhibition because its size is ideally suited for forming strong hydrate interactions. Although the longer PVP16 presents a large contact surface for hydrate interactions, it may be too bulky to achieve adequate surface coverage for complete hydrate growth inhibition.

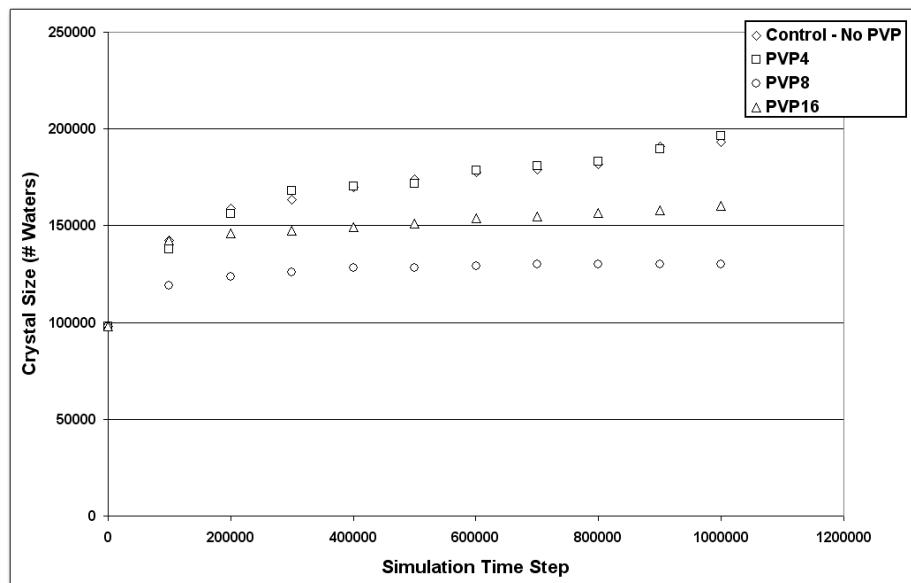


Fig. 6. Gas hydrate growth in the presence of varying length PVP non-crystal molecules

The intermediary effects of PVP16 on hydrate growth suggested that its inhibitory effects might be the most variable of the NCMs examined and prompted us to perform additional experiments on hydrate growth with this NCM. A set of 512 simulations with PVP16 was performed, each one involving a different PVP16-hydrate binding orientation. This was accomplished by rotating PVP16 to each of 16 positions in one rotation axis, 8 positions in a second rotation axis, and finally 4 rotations in a third rotation axis. All 512 simulations began with the same initial 100 000 molecule gas hydrate crystal and operated under the same simulation parameters.

Figure 7 summarizes the sizes of the resulting crystals. While many binding orientations showed little or no effect on hydrate growth, some resulted in substantial growth inhibition. Most surprisingly, others achieved significant growth enhancement. There is a 0.53 correlation between the final crystal size in these simulations and the amount of bound PVP16, suggesting that growth rates may be linked to the quantity of bound PVP16. This relationship is no doubt mitigated by the fact that faster growing crystals will have larger surface areas, providing more binding sites for

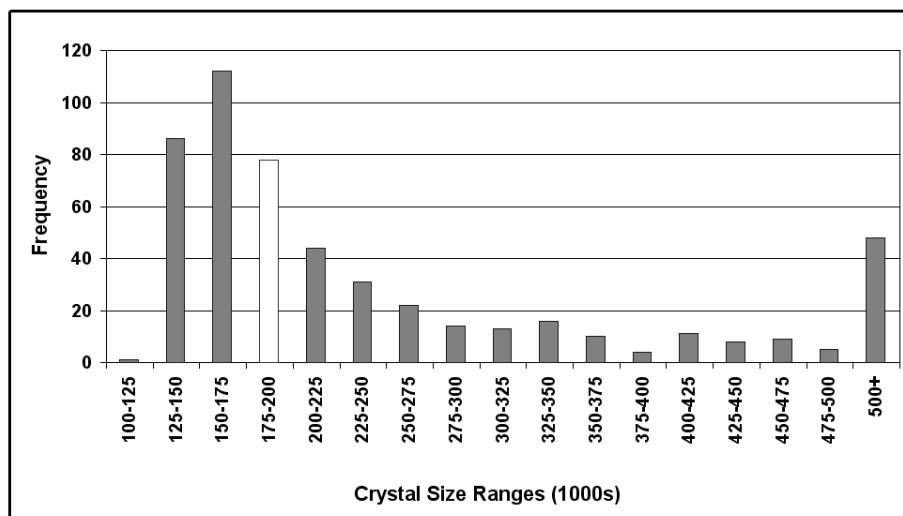


Fig. 7. Number of crystals from the initial screen of 512 simulations involving PVP16 molecules, grouped by crystal size. Control crystals grown without introduced NCMs grow in the 175000-200000 crystal size, indicated by the green bar.

PVP16. Nevertheless, at least in some orientations, the presence of PVP16 appears to nucleate new layers of hydrate growth. Therefore, at a minimum, these computations clearly demonstrate that NCM binding orientation can directly affect crystal growth.

Table 2. Description of pairs of PVP16 molecules with similar orientations but different effects on hydrate growth chosen for detailed study

Molecule	Inhibition/ Nucleation	Crystal Size (waters)	Avg PVP association time, before dissociation (1000's steps)	Avg PVP association time, still bound at simulation end (1000's steps)
180.010.135	Nuc	406461	75.7	409769.0
180.012.135	Inh	163359	14.6	793583.3
270.055.090	Nuc	467272	171.2	481565.7
270.058.090		202540	323.2	529423.4
315.034.090	Nuc	364406	141.3	476201.5
315.036.090	Inh	140251	0.5	884208.9

In an effort to clarify the distinct nucleation and inhibitory abilities of PVP16, we identified pairs of similar PVP16 binding orientations from the initial screens that had opposing effects on hydrate growth. Three pairs were identified (Table 2) with orientations that were separated by 22.5° in a sole rotation axis. For each of these pairs, we

performed an addition series of simulations to identify the precise PVP16 orientations where the PVP molecule changed roles between nucleation and inhibition. Remarkably, these two roles were found to switch in as little as a 2° difference in PVP16 orientations. Subsequent investigations of the shape and volume differences between these PVP16 pairs revealed that in all cases, a single volume position was responsible for the observed nucleation or inhibitory roles of these NCMs. In the case of the 315.034.090 and 315.036.090 PVP16 (with the nomenclature reflecting the three rotations applied to the original PVP16 NCM), which differ by only 2° in the second rotation axis, a single volume difference at one end of these rod-like molecules (Figure 8A) was responsible for either the nucleation or inhibitory roles.

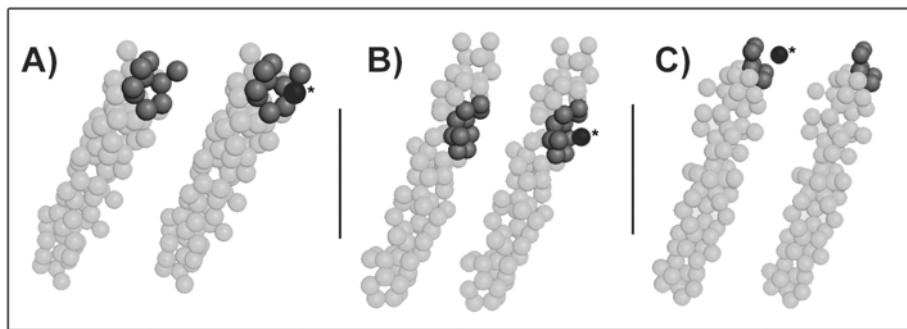


Fig. 8. Pairs of PVP16 NCMs that have nearly identical hydrate binding orientations but dramatically different effects on hydrate growth. In all cases, the NCM volumes differ by a single volume element (dark black spheres identified by an *) that occupies a gas position in hydrate crystals. Capping NCM positions neighbouring these gas positions are shown as grey spheres. Nucleators are on the left in each pair, inhibitors on the right. In both the A) 315.034.090/315.036.090 PVP16 molecules, and B) 180.010.135/180.012.135 PVP16 molecules, the additional volume element results in substantial growth inhibition, while in the C) 270.055.090/270.058.090 PVP16 molecules, the additional volume element is found in the nucleator, which significantly enhances hydrate growth.

An examination of the differences between these two NCMs suggests a mechanism for both nucleation (in the case of 315.034.090 PVP16) and inhibition (for 315.036.090 PVP16) (Figure 9). In the former case, upon hydrate binding, one end of the molecule will partially complete, or cap, a water cage, increasing the likelihood of trapping a gas molecule within the cage long enough for the water cage to develop further. When this NCM subsequently dissociates, we speculate that the majority of the water cage has been assembled, and further crystal growth will complete the cage around the enclosed gas molecule (Figure 9). In short, this orientation of PVP16 appears to promote nucleation by encouraging gas entrapment and subsequent water cage development. In contrast, the 315.036.090 PVP16 molecule, whose volume contains the same partial cage found in the 315.034.090 PVP16 described above, contains one additional volume element that occupies the gas molecule position inside this cage (Figure 8A). Upon hydrate binding, this orientation of the PVP16 molecule appears to achieve inhibition because of two conditions: (i) this extra volume element prevents a gas molecule from

joining the crystal, thus reducing gas uptake by the crystal; and (*ii*) this extra volume becomes ensnared as the water cage encompass this volume element. This latter event locks the inhibitor to the hydrate surface, decreasing the PVP16 dissociation rate and promoting its inhibition abilities (Figure 9). Similar differences between the 180.010.135 PVP16 inhibitor and the closely aligned 180.012.135 PVP16 nucleator appear to also account for their different effects on hydrate growth (Figure 8B). In contrast, the 270.055.090 PVP16 and the 270.058.090 PVP16 molecules appear to affect hydrate growth in a different manner. While the first of these NCMs does not exhibit much influence on crystal growth, the second one clearly acts as a nucleator. Once again, these two PVP16 molecules differ by a sole volume element, and, as before, this volume element occupies a gas position in the crystal upon hydrate binding (Figure 8C). In this case, however, it is the nucleator that contains the extra gas-displacing volume element. One clear distinction between this case and the previous two cases is the size of the water cage cap supplied by the bound NCM. The 270.058.090 PVP16 nucleator supplies the volume of only 6 water molecules to the cage cap, whereas the 315.034.090 PVP16 inhibitor and the 180.010.135 PVP inhibitor provide volume equivalents to 9 and 10 water molecules, respectively, in their cage caps (Figure 8). We speculate that this smaller cage capping may reduce the likelihood of the 270.058.090 PVP16 nucleator becoming trapped within the hydrate surface, preventing it from acting as an inhibitor. Its additional volume in the center part of hydrate cages, however, may encourage cage formation, which will help promote subsequent hydrate growth after the nucleator dissociates.

Though still in their early stages, these investigations into the factors that lead to either accelerated or reduced gas hydrate growth clearly highlight the importance of gas incorporation for hydrate growth. In general, it appears as if good nucleators encourage gas incorporation, while good inhibitors discourage such incorporation. Molecular arrangements that induce cage formation – particularly the shapes and binding orientations of NCMs that cap partially formed cages – appear to be critical for promoting gas uptake during crystal formation (Figure 9). Hydrate growth inhibition, on the other hand, appears to require very tight associations between NCMs and gas hydrates (Table 2): average association times for those NCM inhibitors that remain bound at the conclusion of their simulations are almost double those for NCM nucleators. Moreover, the average association times for NCM inhibitors that dissociate during simulations are much lower than for nucleators. Taken together, this suggests that inhibitor associations are either short-lived, reflecting poor binding complementarity between inhibitor and crystal surface, or very long lasting. Indeed, in many of the simulations involving docking orientations of PVP16 that resulted in significant reductions in hydrate growth rates, the PVP16 molecules appear to bind irreversibly, suggesting that growth inhibition is accomplished by physical surface coverage. As described in Figure 9, inhibitors that occupy the central portions of hydrate cages appear to have increased binding efficacies, as the cages grow up around the bound inhibitors, helping to anchor them to the crystal surfaces.

In summary, these simulations are preliminary, and much work remains to be done. Undeniably, this work stimulates more questions than it answers: is irreversible binding absolutely necessary for inhibition, or can inhibition be achieved while maintaining a basal level of inhibitor dissociation? Which specific features of the energetics promotes gas hydrate nucleation and inhibition? Can the results of simulation

studies assist in the development of actual hydrate inhibitors and nucleators? The next phase of our research will address these and other related questions. We hope that answers to even some of these will lead to a better understanding of the mechanisms of gas hydrate growth and growth inhibition.

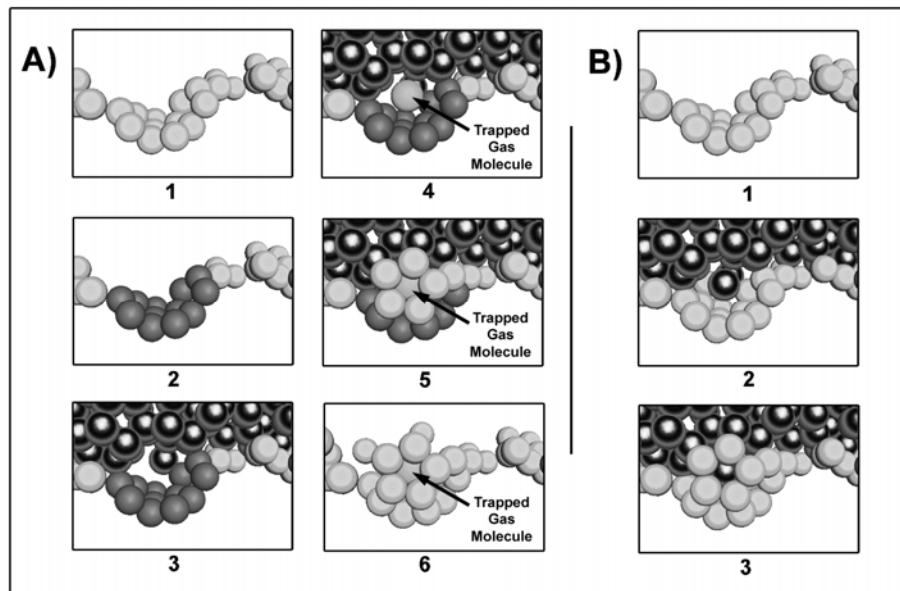


Fig. 9. Proposed mechanisms for PVP-mediated alterations to gas hydrate growth rates. A) Enhanced hydrate growth. (1) The water molecules on the hydrate surface are shown as light grey spheres, a subset of which are identified in (2) (dark grey spheres) that form the bottom portion of a water cage. (3) A PVP16 molecule binds to the surface (black spheres), capping the partially formed cage. (4) Subsequently, a gas molecule enters the cage (light grey sphere, as indicated). This gas molecule will then encourage the completion of the water cage (5). Finally, following PVP16 dissociation, the water cage is completed, and a new layer of hydrate growth ensues. B) Inhibited hydrate growth. (1) The same hydrate surface is shown as in growth enhancement. (2) A PVP16 molecule that includes volume which fills the centre of a water cage binds to the surface, thereby preventing a gas molecule from associating to the crystal surface. (3) Water molecules are encouraged to grow around this additional PVP16 volume element, trapping the PVP16 molecule on the hydrate surface and leading to crystal growth inhibition.

Acknowledgments. This research is supported by the High Performance Computing Virtual Laboratory in Kingston, Canada and by a NSERC (Canada) Strategic and Discovery grants to VKW. We thank Dr. S. Akl for encouragement. BW is a recipient of an NSERC graduate studentship, and the Sun Microsystems of Canada Scholarship in Computational Sciences and Engineering. ZJ is a Canada Research Chair in Structural Biology and VKW holds a Queen's Research Chair.

References

1. Kvamme, B., Kuznetsova, T., Aasoldsen, K.: Molecular dynamics simulations for selection of kinetic hydrate inhibitors. *J. Mol. Graphics Modelling* 23, 524–536 (2005)
2. Sloan Jr., E.D.: Fundamental principles and applications of natural gas hydrates. *Nature* 426, 353–362 (2003)
3. Gao, S., House, W., Chapman, W.G.: Detecting Gas Hydrate Behavior in Crude Oil Using NMR. *J. Phys. Chem. B* 110, 6549–6552 (2006)
4. Han, F.X., Lindner, J.S., Wang, C.: Making carbon sequestration a paying proposition. *Naturwissenschaften* 94, 170–182 (2007)
5. Linga, P., Kumar, R., Englezos, P.: The clathrate hydrate process for post and pre-combustion capture of carbon dioxide. *J. Hazard Mater.* 149, 625–629 (2007)
6. Strobel, T.A., Kim, Y., Andrews, G.S., Ferrell III, J.R., Koh, C.A., Herring, A.M., Sloan Jr., E.D.: Chemical-clathrate hybrid hydrogen storage: storage in both guest and host. *J. Am. Chem. Soc.* 130, 14975–14977 (2008)
7. Sloan Jr., E.D.: Clathrate Hydrates of Natural Gases, 3rd edn. Crc Press Llc, New York (2007)
8. Konstantin, A.U., Lu, H., Enright, G.D., Ratcliffe, C.I., Ripmeester, J.A., Chapman, N.R., Riedel, M., Spence, G.: Single Crystals of Naturally Occurring Gas Hydrates: The Structures of Methane and Mixed Hydrocarbon Hydrates. *Angew. Chem. Int. Ed.* 46, 8220–8222 (2007)
9. Schicks, J.M., Naumann, R., Erzinger, J., Hester, K.C., Hoh, C.A., Sloan Jr., E.D.: Phase Transitions in Mixed Gas Hydrates: Experimental Observations versus Calculated Data. *J. Phys. Chem. B* 110, 11468–11474 (2006)
10. Patchkovskii, S., Tsu, J.S.: Thermodynamic stability of hydrogen clathrates. *Proc. Natl. Acad. Sci. USA* 100, 14645–14650 (2003)
11. Li, J., Liang, D., Guo, K., Wang, R.: The influence of additives and metal rods on the nucleation and growth of gas hydrates. *J. Colloid Interface Sci.* 283, 223–230 (2005)
12. Koh, C.A.: Towards a fundamental understanding of natural gas hydrates. *Chem. Soc. Rev.* 31, 157–167 (2002)
13. Anderson, B.J., Tester, J.W., Borghi, G.P., Trout, B.L.: Properties of inhibitors of methane hydrate formation via molecular dynamics simulations. *J. Am. Chem. Soc.* 127, 17852–17862 (2005)
14. Larsen, R., Knight, C.A., Rider, K.T., Sloan Jr., E.D.: Growth and Inhibition of Ethylene Oxide Clathrate Hydrate. *Ann. N.Y. Acad. Sci* 912, 441–451 (2006)
15. Kumar, R., Lee, J.D., Song, M., Englezos, P.: Kinetic inhibitor effects on methane/propane clathrate hydrate crystal growth at the gas/water and water/n-heptane interfaces. *J. Crystal Growth* 310, 1154–1166 (2008)
16. Moon, C., Taylor, P.C., Rodger, P.M.: Molecular Dynamics Study of Gas Hydrate Formation. *J. Am. Chem. Soc.* 125, 4706–4707 (2003)
17. Alavi, S., Ripmeester, J.A., Klug, D.D.: Molecular-dynamic Simulations of Binary Structure II Hydrogen and Tetrahydrofuran Clathrates. *J. Chem. Phys.* 124, 14704–14709 (2006)
18. Alavi, S., Ripmeester, J.A., Klug, D.D.: Molecular Dynamics Study of the Stability of Methane Structure H Clathrate Hydrates. *J. Chem. Phys.* 126, 124708–124713 (2007)

19. Moon, C., Hawtin, R.W., Rodger, P.M.: Nucleation and control of clathrate hydrates: insights from simulation. *Faraday Discuss.* 136, 367–382 (2007)
20. Wathen, B., Kuiper, M., Walker, V.K., Jia, Z.: A New Model for Simulating 3-D Crystal Growth and its Application to the Study of Antifreeze Proteins. *J. Am. Chem. Soc.* 125, 729–737 (2003)
21. Wathen, B., Kuiper, M., Walker, V.K., Jia, Z.: New Simulation Model of Multicomponent Crystal Growth and Inhibition. *Chem. Eur. J.* 10, 1598–1605 (2004)
22. Gilmer, G.: Computer Models of Crystal Growth. *Science* 208, 355–363 (1980)

Nonlinear Time-Dependent Density Functional Theory Studies of Ionization in CO₂ and N₂ by Intense Laser Pulses and Molecular Orbital Reconstruction

Emmanuel Penka Fowe and André Dieter Bandrauk

Laboratoire de Chimie Théorique, Faculté des Sciences, Université de Sherbrooke,
Sherbrooke, Québec, J1K 2R1, Canada
Andre.Dieter.Bandrauk@USherbrooke.ca

Abstract. Time-dependent density functional theory, TDDFT, studies of the ionization of CO₂ and N₂, by intense laser pulses peak intensities (3.50×10^{14} , 1.40×10^{15} , 2.99×10^{15} and 5.59×10^{15} W/cm²) at 800 nm ($\omega = 0.0584$ a.u.) are presented in the nonlinear nonperturbative regime using the LB94 potential which reproduces the ionization potential of our systems more accurately, without significant increase in computational costs over a local-density approximation. Special emphasis is placed on elucidating molecular orbital (MO) orientation and various peak intensities effects on the ionization processes. The results reveal that molecular orbital ionizations are strongly sensitive to their symmetry (MO shape), the induced dipole coupling between molecular orbitals, and the laser intensities. Notably, we found that with a proper choice of the laser intensity (3.5×10^{14} W/cm²), the sensitivity is strong enough so that the nature and symmetry of the highest occupied molecular orbital can be directly probed and visualized from the angular dependence of laser induced ionization. At higher intensities, ionization is found to occur also from inner orbitals, thus complicating imaging of simple orbitals.

Keywords: TDDFT, molecular orbital ionization probability, total molecular ionization yield, molecular orbital symmetry, lasers intensities.

1 Introduction

In recent years, interest the interaction of atoms and molecules with intense laser pulses has grown significantly, especially since the availability of a single isolated attosecond pulse [1] has paved the way to what has been called attoscience [2]. In particular in molecular ionization studies [3], accurate experimental techniques [4] for molecular orientation and alignment have been developed. As a result, tomographic [5] imaging of molecular orbitals has emerged as one of the active topics in strong field atomic and molecular physics. While experimental efforts for better understanding of the tomographic imaging processes are growing fast, theoretical investigations are still limited, even though computational power is increasing rapidly. It is worth noting that the theoretical description of the tomographic imaging of any molecular orbital needs the explicit treatment of the electron-electron interaction -- a very difficult task still far from

being solved, in contrast with the one electron systems (H_2^+ , HeH^{++} and H_3^{++}) which have been well studied in our group (Kamta et al.[6]) by solving exactly the time-dependent Schrödinger equation, TDSE. However, it is possible to reformulate the problem of calculating the multi electron-electron interaction in molecule by means of the density functional theory [7,8] (DFT), which has become one of the most widely used electronic structure methods because of its ability to provide an accurate description at a bearable computational cost. Subsequently, it has been shown [9] that general time-dependent density functional theory, TDDFT, as opposed to its linear response form [10], provides a framework where the dynamics of several interacting electrons driven by strong external fields can be studied in a computationally efficient manner in the nonlinear perturbative regime. However, the drawback of the DFT or TDDFT approach is that the key parameter, the exchange-correlation (xc) functional is explicitly unknown. Commonly used approximations for functionals such as the local density approximation LDA have a notorious failure which has been identified as the self-interaction (SI) error resulting from the incomplete cancellation of an interaction of an electron with itself. Typically, self-interaction leads to incorrect dissociation limits [11], underestimation of energy barriers to chemical reactions [12], the ionization potential (IP), and a wrong asymptotic behaviour of the exchange-correlation potential [13].

The motivation of this article is as follows. We extend the TDDFT, with proper long range potentials, to the tomographic imaging of molecular orbitals of CO_2 and N_2 in a strong laser field from the angular dependent molecular ionization, with a special emphasis placed on elucidating molecular orbital orientation effects, non-linear response of each individual Kohn Sham [8] (KS) orbital and laser peak intensity effects on the ionization processes. In our previous study [14] on the tomographic imaging of molecular orbitals of CO_2 , we found that molecular orbital ionizations are strongly sensitive to their symmetry, the laser intensities and that the highest occupied molecular orbital (HOMO) is not necessarily the dominant channel responding to the strong-field molecular ionization. Most remarkably, we found that with a proper choice of the laser intensity ($3.5 \times 10^{14} \text{ W/cm}^2$), the sensitivity was strong enough such that the nature and symmetry of the highest occupied molecular orbital was directly probed and visualized from the angular dependence of laser induced ionization. However, our calculations were performed using the LDA approximation. Therefore, we felt it was challenging to analyse in more detail the performances of different exchange-correlation potentials for the calculation of the IP, and to assess which is the best choice with respect to experiment. It is important to underline that all these choices are made in order to optimize the time independent description of the system at the KS level, but the electron dynamics will be treated with the time-dependent extension of the density functional theory, i.e., TDDFT in the nonlinear nonperturbative regime. Using chosen functionals, we will compute the molecular and orbital ionization probability, and then we will study the response in time of N_2 and CO_2 , to different laser pulses intensities having a wavelength of 800 nm. We have chosen these two molecules for two main reasons: there are extensive experimental data available [15,16]; their electronic valence nature is rather similar to each other, the highest occupied molecular orbitals (HOMO), HOMO-1 and HOMO-2 are respectively pi, sigma, pi type for CO_2 , and sigma, pi and sigma types for N_2 . Thus, we investigate how the MO shapes affect the molecular ionization according to the laser orientations and intensities. The organization of this article is as follows. First we briefly describe the

TDDFT formalism and give the procedure for calculating ionization rates. The second section is devoted to a short review of the calculation of the IP, followed then by the computational details. Results and discussions are given in the last section. The paper is ended by some concluding remarks.

2 Theory

2.1 Nonperturbative TDDFT

The TDDFT method provides the most detailed and feasible *ab-initio* approach for tackling many-body problems. Density functional theory as first introduced by Hohenberg and Kohn⁷, and extended by Kohn and Sham [17] is based on the existence of an exact mapping between one-particle density and external potentials. This leads to the density of the interacting system being obtained from the density of an auxiliary system of non-interacting particles moving in an effective local single-particle potential. A time dependent generalisation of DFT, TDDFT, was provided by Runge and Gross [18], showing that there is a one-to-one correspondence between the external (time dependent) potential, $v_{ext}(r,t)$, and the time dependent one-electron density, $n(r,t)$, for many-body systems evolving from a fixed initial state. The time dependent electronic density is written as: [19]

$$n(r,t) = \sum_{\sigma=\uparrow,\downarrow} n_{\sigma}(r,t) = \sum_{\sigma=\uparrow,\downarrow} \sum_i^{N_{\sigma}} |\psi_{i\sigma}(r,t)|^2, \quad (1)$$

where, $N_{\sigma} = N_{\downarrow}, N_{\uparrow}$ is the number of occupied orbital for a giving spin σ , and $\psi_{i\sigma}(r,t)$ is the occupied orbital obtained through the time dependent KS equations (in a.u. where $\hbar=m_e=e=1$),

$$i \frac{\partial}{\partial t} \psi_{i\sigma}(r,t) = \left[-\frac{1}{2} \nabla^2 + v_{eff}(r,t) \right] \psi_{i\sigma}(r,t), \quad (2)$$

where

$$v_{eff}(r,t) = v_{ext}(r,t) + v_h(r,t) + v_{xc,\sigma}(r,t). \quad (3)$$

The first term is the external potential, from the interaction of the electron with an external laser field and the nuclei, while the second term accounts for the classical Hartree electrostatic interaction between electrons, and the third, the exchange-correlation potential includes all non-trivial many body effects, and has an extremely complex (and essentially unknown) functional [9, 17] dependence on the density. This dependence is non-local both in space and in time, thus, the quality of numerical solutions of Eq.(2) will depend on the quality of the approximation to the *xc* potential used.

2.2 Ionization Potential

When an intense electric field is applied on a molecule, it tries to make the electrons free by ionization; thus, the highest IP or barriers will hinder this electronic movement much more than the lower barriers. Therefore, the choice of an approach which correctly reproduces the experimental IP is decisive for our analysis. To compute this process, DFT and Hartree Fock (HF) approaches were used for comparison. For HF, the IP is computed using the simplest treatment based on Koopmans' theorem [20], which states that the ionization potential is given by the negative of the restricted (closed-shell) Hartree Fock orbital energy, $-\epsilon_i$, calculated in the neutral system. This approach ignores the relaxation of the molecular orbitals after the ionization, however, this effect tends to be cancelled by the absence of the electron correlation in the HF wave function [21]. In DFT, Koopmans' theorem is not explicitly applicable, but the eigenvalue of the highest KS orbital has been proven by Janak [22] to be the IP if the functional is exact. Several functionals have been tested, in particular the local-density approximation (LDA) which has been widely used in a variety of fields (Chemistry, Physics) due to its simplicity and applicability to various systems with relatively less computational cost. However, LDA suffers as we mentioned in section 1) from the wrong asymptotic behaviour originating from the incomplete cancellation of the self-interactions, and also, the correlation energy of an electronic system in LDA is based on the homogeneous gas system; thus the exchange-correlation energy due to the inhomogeneities in the electronic charge density are ignored. For these reasons we have move beyond the LDA, notably through the addition of gradient corrections (GGA) to incorporate longer range gradient effects in which the exchange correlation energy functional explicitly depends on the gradient of a density as well as the density itself. However, GGA is also unable to reproduce the correct asymptotic behavior of the exchange-correlation potential. Furthermore, the hybrid approximations where some amount of exact HF exchange [23] is mixed with exchange-correlation functionals from LDA and GGA has also been tested [25].

We have used the Van Leeuwen and Baerends [25] potential (LB94) which introduces a gradient correction to the LDA exchange correlation potential so as to reproduce the correct asymptotic behaviour. A convenience of the LB94 potential lies in its explicit dependency on the local density and the gradient of the local density. This functional can yield good estimation of the ionization potential for atoms and small molecules from the highest occupied orbital energies [26], especially in the framework of the TDDFT excitation energy [27] (linear response). A method to completely remove the SI was proposed by Perdew and Zunger [28] using orbital dependent functional, similar to the HF. Unfortunately, the equations that emerge depend on each spin-orbital in the system and the solutions of the Kohn–Sham equations are complicated. This latter problem is usually overcome by means of the optimized effective potential (OEP) method [29], and the approximation of Krieger, Li, and Iafrate [30, 31] (KLI) which is widely used as a good alternative to obtain a local potential for functionals that depend explicitly on the spin–orbitals [32] such as the SI correction approximation. The KLI-OEP method offers the advantage of using canonical equations.

3 Computational Details

In the present work, the TDDFT KS equation, Eq.(2), was discretized in space using finite-difference (FD) grid techniques combined with the Troullier – Martins pseudo-potential [33] and the resulting computer code was parallelized to run on a massively parallel processor at RQCHP [34]. We have placed a CO₂ and N₂ molecule in a large three-dimensional (3D) cubic grid cell (atomic units a.u., are used) of dimension a = 64 a.u. (1a.u.=0.0529nm). The uniform FD grid spacing, Δa = 0.26 a.u., was used for numerical integration and the convergence of the calculations was checked against results making use of a smaller grid spacing. The external potential created by an intense laser field is taken to be an oscillating electric field with a cosine envelope of the form:

$$v_{ext}(r,t) = \mathbf{r} \cdot \mathbf{E}(t); \quad E(t) = E_o \cos^2\left(\frac{\pi(t - 2\tau - t_o)}{2\tau}\right) \sin(\omega t). \quad (4)$$

E_o is the maximum field strength, τ is half of the total pulse duration, $\omega = 0.0584$ a.u corresponding to λ=800nm and t_o is chosen such that |t-t_o| < τ. The laser field linear polarization is set parallel to the molecular axis (z) and we have defined, θ, as the molecular orientation angle. The total area, $\int_{-\infty}^{+\infty} E(t) dt = o$, ensures no spurious static effects [35].

The time-dependent equations are solved using the classic Crank-Nicholson scheme [36]. To absorb the electrons reflecting spuriously at the boundary of the box, we included an imaginary potential [37] in the Hamiltonian of the system. This acts as an absorber for electrons at the boundaries of the grid. Thus, once the time-dependent wave functions and the time dependent electron density were obtained, we have easily calculated the time-dependent ionization probability, $P_{i,\sigma}$, of an individual spin-orbital according to:

$$P_{i,\sigma} = 1 - N_{i,\sigma}(t), \quad (5)$$

where

$$N_{i,\sigma}(t) = \langle \psi_{i,\sigma}(r,t) | \psi_{i,\sigma}(r,t) \rangle, \quad (6)$$

is the time dependent population (survival probability) of the i, σ -th spin-orbital. The number of the electrons inside the grid of volume V is given as:

$$N_{bound}(t) = \int_V d^3r n(r,t) \quad (7)$$

and the normalised remaining electron population as:

$$\beta(t) = \frac{\int_V d^3r n(r,t)}{\int_V d^3r n(r,0)} = e^{-\Gamma t}, \quad (8)$$

Γ is the ionization rate and $n(r,0) = 2 \sum_i^{N_\sigma} |\psi_{i\sigma}(r,t)|^2$ is the total number of electrons in the initial state (without external field). The ionization yield probability, $\gamma(t_f)$, computed at the final time, t_f , of the propagation is then obtained from the following difference:

$$\gamma(t_f) = 1 - \beta(t_f), \quad (9)$$

and the KS orbital ionization probability, $\delta(t_f)$, is deduced as:

$$\delta(t_f) = 1 - N_{i,\sigma}(t_f).$$

4 Results and Discussions

4.1 Ionization Potential, IP

The computed values of the IP of the CO₂ and the N₂ molecule at various ab initio theory levels are presented in Table 1.

Table 1. Ionization potential (eV) computed by the Hartree-Fock and various DFT methods as IP=-ε_{HOMO}

	HF	DFT						Expt ³⁹
		LDA	OEP	GGA		Hybrid		
	VWN ⁴⁰	OEP-KLI ³¹	PW91 ⁴¹	LB94 ²⁵	B3LYP ²³	O3LYP ⁴²		
CO ₂	14.69	9.63	16.04	10.07	15.23	10.24	10.38	13.77
N ₂	16.59	10.89	16.87	10.91	15.87	12.35	12.19	15.58

Table 1 shows that the values of the IP calculated for the two molecules with the HF approach are higher (around 7%) than the experimental data. This deviation is most related to the absence of the correlation in the HF approach. Using DFT, the expected underestimation of the ionization potential at the LDA level is clearly noticeable. Computed values are approximately 43% lower than the experimental ones. This behavior doesn't improve upon the correction of the long range gradient density effect brought through the GGA functional (PW91, BLYP). The reason of this large discrepancy lies, as we mentioned in the previous theoretical section 2), in the wrong asymptotic behaviour of the LDA and GGA exchange-correlation potential originates from the SI, as opposed to the HF approach (SI free eliminated by the exact exchange).

Results using partially corrected SI through the hybrid functional (B3LYP, O3LYP) moderately improve the calculated value of the IP, the relative error drops from 43% with the LDA and GGA to 25 %. Subsequently, the use of the KLI-OEP, with the LDA correlation considerably improves the quality of our results. According to our calculations, the deviation from the experimental data is 8% higher for the two

molecules. This observed discrepancy may arise from the KLI approximation itself and also from the failure of the LDA treatment of the electronic correlation due to the presence of multiple bonds in CO₂ and N₂. The most important finding is that the LB94 potential performs much better for IP; the global agreement with experiment looks satisfactory which is within 5% of error. We have also tested the computation of the IP with the Franck–Condon principle which states that the most probable ionizing transition will be that in which the positions and moments of the nuclei are fixed. Practically, this is done by the calculation of the vertical ionization energy, i.e. a proper subtraction of the total energies for the neutral and the cation obtained at the equilibrium geometry of the neutral. This includes therefore relaxation in the cation. Except for HF within 8% of error, we found that results are close to the experimental value within 6% of error, and are not sensitive to different functionals.

4.2 Molecular Orbital of CO₂ and N₂

The relevant information of the ground state geometry of CO₂ and N₂ is that the two molecules are linear and the experimental [38] bond length used for the two molecules are 1.162 Å for CO and 1.041 Å for NN.

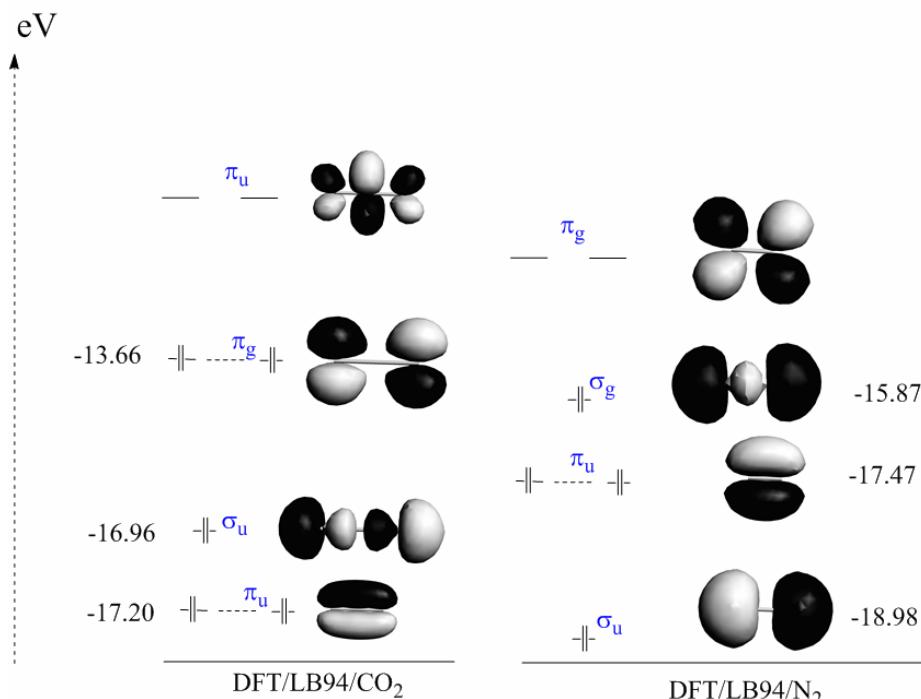


Fig. 1. DFT/LB94 images of the molecular orbitals of CO₂ and N₂. Only the three highest occupied molecular orbitals, HOMO, HOMO-1 and HOMO-2 are plotted.

The analysis of the electronic structure from the DFT-KS calculation is depicted in Fig.1. The three highest occupied KS molecular orbitals with their phases are presented. The nodal properties which correctly reflect the *g* and *u* symmetry character of the orbitals are visible. One finds that the highest occupied molecular orbital, HOMO, is respectively the anti-bonding $1\pi_g$ for CO₂ formed of pi lobes lying perpendicular to the inter-nuclear axis and located on the oxygen atoms and the bonding σ_g for N₂ formed mostly by the p_z orbital overlap between nitrogen atoms. The lowest unoccupied molecular orbital LUMO, is also shown for each molecule. The assigned electronic configuration follows from the orbital energies is KK(π_u)⁴(σ_u)²(π_g)⁴ for CO₂ and KK(σ_u)²(π_u)⁴(σ_g)² for N₂, where KK is the configurations of inner shell orbitals.

4.3 Orbital Ionization

To gain a better understanding of the ionization dynamics, we have investigated the time evolution of the KS orbital following Eq.(6). The laser (electric) field is assumed to be either parallel ($\theta = 0^\circ$) or perpendicular ($\theta = 90^\circ$) to the molecular axis, and the internuclear distance of each molecule is set at its experimental value.

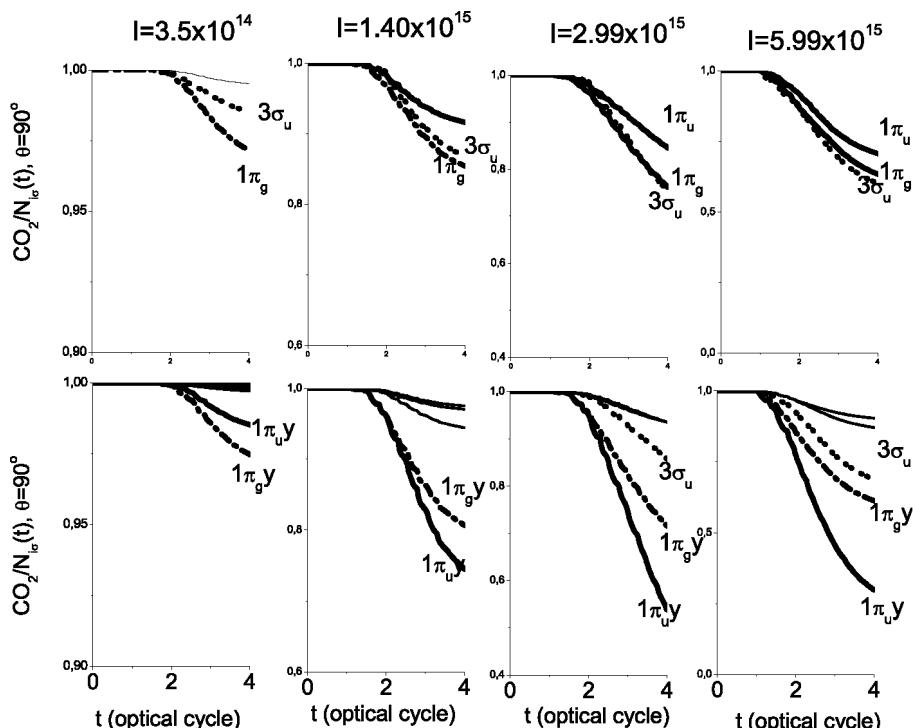


Fig. 2. KS orbital population, $N(t)$, as a function of the laser intensities I (W/cm²); only the relevant KS orbitals which possess an important response to the laser fields are shown with their label. a) top: the laser is parallel to the CO₂ axis; b) bottom: the laser is perpendicular to the CO₂ axis.

Results for four laser intensities 3.50×10^{14} , 1.40×10^{15} , 2.99×10^{15} and 5.59×10^{15} W/cm² are displayed in Fig.2 for CO₂ and Fig.3 for N₂. Calculations have been done using four optical cycle pulse duration. One finds that when $\theta=0^\circ$, for CO₂ molecule (Fig.2), the two components of each π orbital have the same behaviour for symmetry reason. At lower intensities, 3.50×10^{14} , 1.40×10^{15} W/cm², it emerges that the HOMO $1\pi_g$ molecular orbital shows as expected due to its lower IP, the dominant response to the laser field and is followed by the inner $3\sigma_u$ MO whose ionization increases steadily with the increasing laser intensities. For the last two higher intensities, 2.99×10^{15} , 5.59×10^{15} W/cm², it is clear that the HOMO does not show the dominant response to the field. Instead, our DFT/LB94 calculation shows that the ionization of the inner $3\sigma_u$ molecular orbital is so important that it exceeds that of the HOMO and thus presents the dominant response to the field. One observes also that the ionization of the inner bonding $1\pi_u$ MO is increasing as a function of laser intensity. When $\theta=90^\circ$, the symmetry behaviour of the different components of the π orbital is broken (degeneracy is removed) and renamed (e.g. π components are renamed π_gx and π_gy). At the lowest laser intensity 3.50×10^{14} W/cm², the HOMO $1\pi_gy$ shows the dominant response to the field followed by the $1\pi_{uy}$ bonding HOMO-2. As the laser intensity is increasing, one sees remarkably that the π_u bonding HOMO-2 ionizes faster than the π_g anti-bonding HOMO (we use the notation HOMO-1, and HOMO-2 for first and second HOMO below the HOMO).

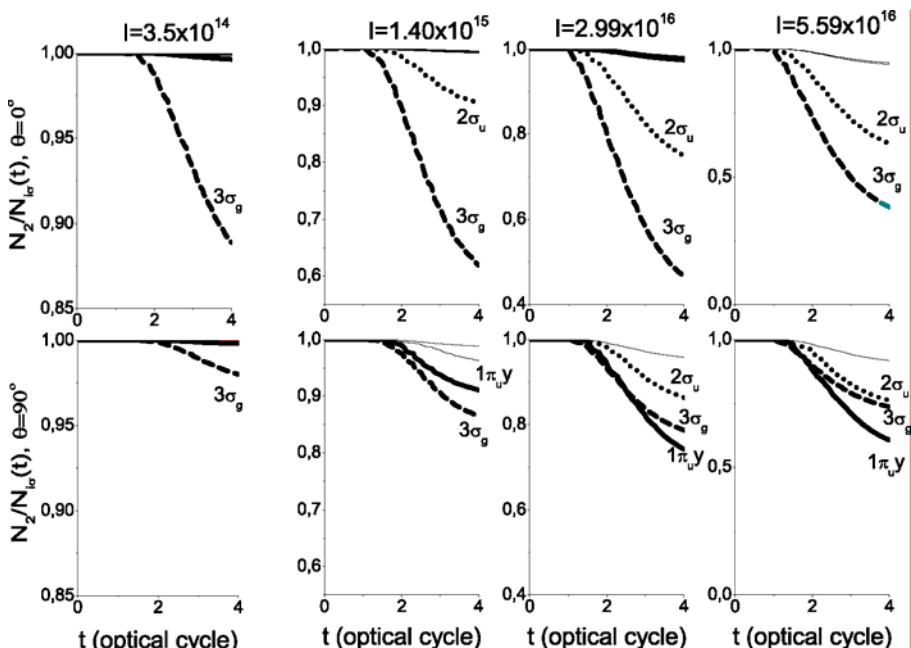


Fig. 3. KS orbital population, $N(t)$, as a function of the laser intensities I (W/cm²); only the relevant KS orbitals which possess an important response to the laser fields are shown with their label. a) top: the laser is parallel to the N₂ axis; b) bottom: the laser is perpendicular to the N₂ axis.

For N₂, (Fig.3), one observes that for $\theta=0^\circ$, the bonding valence orbital having symmetry $3\sigma_g$ shows the dominant response to the field followed by the $2\sigma_u$ anti-bonding HOMO-2, while the ionization of the π_u HOMO-1 is not evident when the laser field is parallel to the molecular axis although the IP of $1\pi_u$ electrons is around 1.5 eV less than that of the $2\sigma_u$ electrons. This trend remains unchanged for the different laser intensities considered in this work. The increasing of the ionization of sigma type orbitals with the laser intensity, results from both the symmetry allowed coupling between them, and the proper orientation of their shape along the laser polarisation. By analysing the case $\theta=90^\circ$, one finds again that for the first lower lasers intensities (3.50×10^{14} W/cm² and 1.40×10^{15} W/cm²) the HOMO $3\sigma_g$ ionizes fast, and for higher laser intensities, the HOMO-1 $1\pi_u$ presents the dominance response to the laser, similar to the CO₂ results.

With regard to our results, one can conclude that the orientation or the shape of the orbital plays an important role in the ionization process. At lower intensities, the ionization probability is more sensitive to the ionization potential than to the geometry of the wave function, i.e. the anisotropy of the ionization is more significant at smaller laser intensities where the expected higher valence orbital is more active, whereas at higher intensity, the inner electrons whose orbital orientation is parallel to the

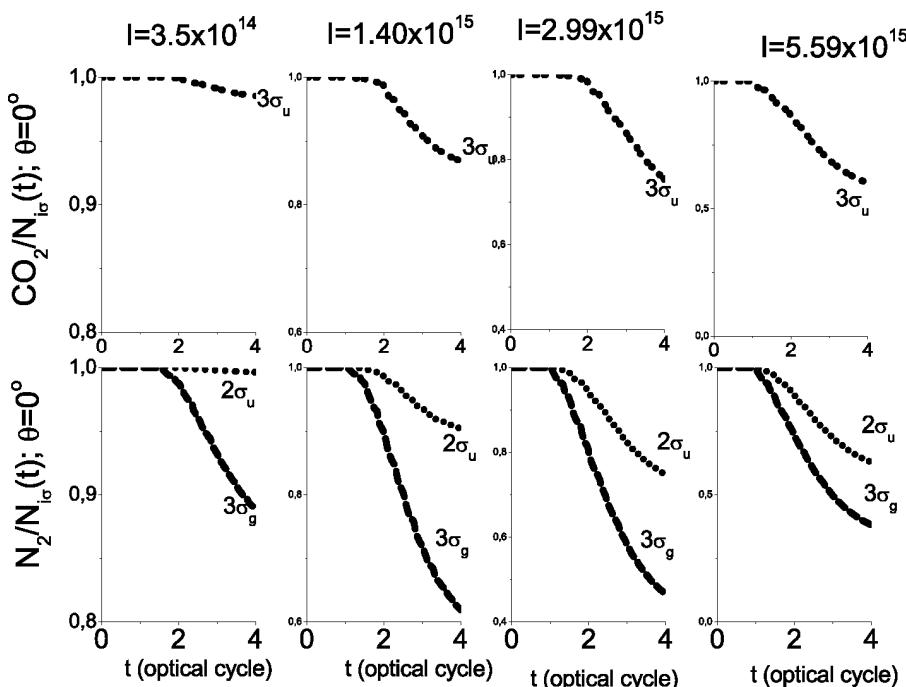


Fig. 4. KS orbital population, $N(t)$, as a function of the laser intensities I (W/cm²); only the sigma orbitals are shown with their label. a) top: CO₂ axis; b) bottom: N₂. The laser is parallel to the molecular axis.

polarization direction of the laser field ionizes faster. One also notes that high laser intensities tend to enhance the symmetry allowed MO dipole coupling. To check that, we plot in Fig. 4 the orbital population of the KS orbital of sigma types for CO_2 ($3\sigma_u$) and N_2 ($3\sigma_g$, $2\sigma_u$) with $\theta = 0^\circ$.

One can easily notice that for N_2 the slope of the KS orbital ionization is higher compare to that of CO_2 , and this trend is increasing with respect to the lasers intensities. We attribute that to the coupling between the two sigma MOs of N_2 . In Fig.5, we repeat the same operation using the KS orbital of pi types for CO_2 (π_g , π_u) and N_2 (π_u) when $\theta = 90^\circ$.

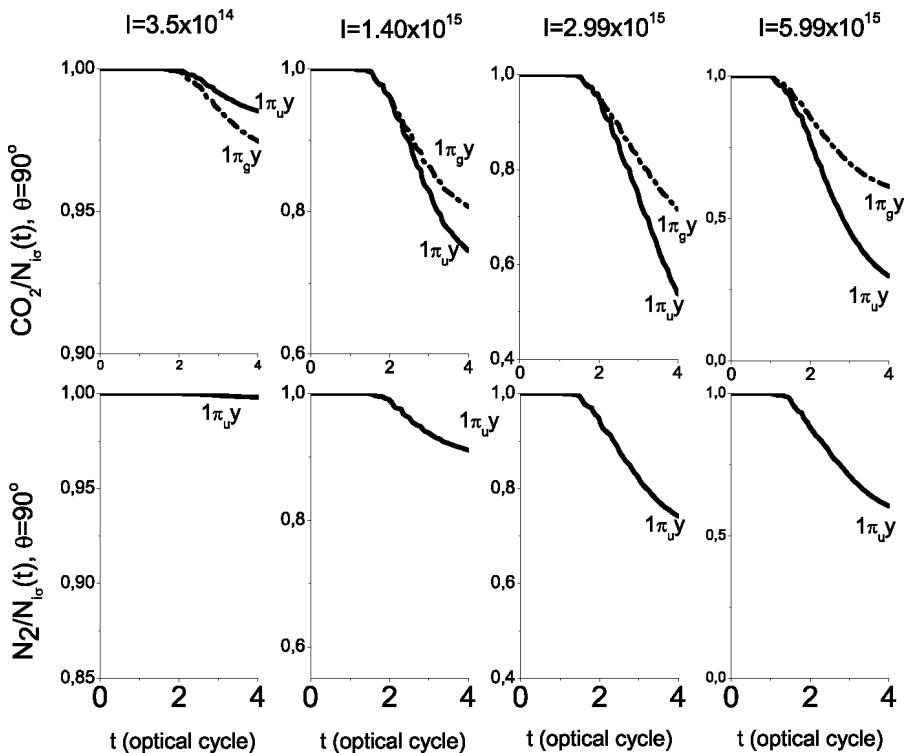
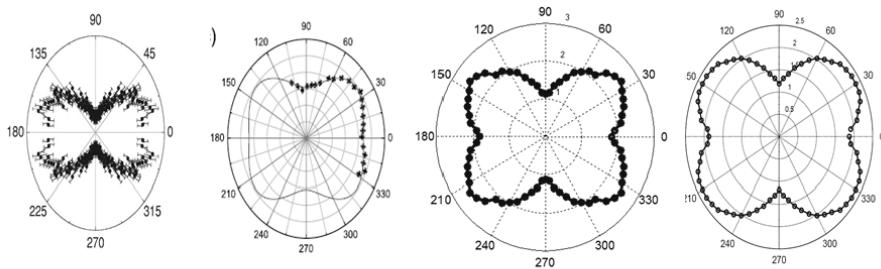


Fig. 5. KS orbital population, $N(t)$, as a function of the laser intensities I (W/cm²); only the sigma orbitals are shown with their label. a) top: CO_2 ; b) bottom: N_2 . The laser is perpendicular to the molecular axis.

One sees that MOs of the CO_2 molecule are ionized faster than the single N_2 MO, resulting most probably from the induced dipole coupling between the π MO of CO_2 . More surprisingly, one notes for CO_2 that the bonding MO shows the dominant response to the laser field even if its KS eigenvalue is smaller than its homolog anti-bonding MO.

4.4 Tomographic Imaging of the HOMO of CO₂ and N₂

Regarding our data on Fig.2 and Fig.3, one easily notes that at lower intensity, the HOMO orbital remains the most influenced by the laser field independently of polarization axis. To analyse the details of the angular-dependent ionization rate of CO₂, the corresponding ionization probability, $\gamma(t_f)$, Eq.(9) vs the orientation angle, θ , is plotted (3.50×10^{14} W/cm² for CO₂, 7.88×10^{14} W/cm² for N₂) in polar coordinates in Fig.6 for CO₂ and Fig.7 for N₂ using 6 optical cycles. Experimental data are also shown for comparison. The calculations were performed with, θ , varying from 0° to 90° (with a 5° angle step) and were reflected to the rest of the trigonometric circle to give a complete picture.



Analser et al. Villeneuve et al. TDDFT/LB94 TDDFT/LDA

Fig. 6. Computed and measured ionization yield, $\gamma(t_f)$, as a function of the angle, θ , between the CO₂ molecule and the laser beam: a) experimental angular [16] distributions for correlated (CO⁺, O⁺) ion pairs produced by dissociation of CO₂⁺ by (8-fs, 800-nm central wavelength and 1×10^{14} W/cm²) laser pulses; b) Data from Villeneuve et al. [15] (1.1×10^{14} W/cm²); c) simulated angular distributions of the first ionization [16] from ADK theory; d) results from the TDDFT of this work.

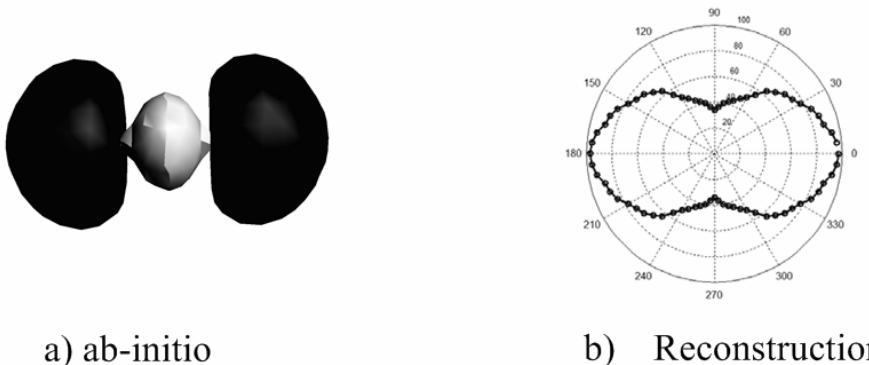


Fig. 7. Computed and measured ionization yield, $\gamma(t_f)$, as a function of the angle, θ , between the N₂ molecule and the laser beam: a) HOMO of N₂ from ab-initio calculation; b) results from the TDDFT reconstruction.

For CO₂, it appears from our result that the ionization rate peaks somewhere between 30° and 40°, and shows minima at 0° and 90° in agreement with our finding in the previous work [14]. This is because the HOMO has a nodal plane containing (or perpendicular) the molecular axis, giving rise to a low probability for ionization when aligned parallel (or perpendicular) to the laser electric field and this is corroborated by exact TDSE calculations in H₂⁺ [6].

For N₂, as expected, the maximum of the ionization rates peaks at 0° due to the HOMO 3σ_g KS orbital. Our plot exhibits a butterfly pattern which is characteristic of the 1π_g orbital for CO₂ (Fig.5) and the shape of the bonding HOMO for N₂. The nonzero ionization at θ=0° and θ=90° for CO₂ reflects contributions from the inner orbitals.

5 Conclusion

Current laser technology allows for the generation of ultrashort intense laser pulses in the femtosecond (10⁻¹⁵s) time regime. This has lead to a new application in imaging molecular phenomena, tomography of molecular orbitals. We have presented in the present work highly parallel numerical simulations to establish on firm theoretical and numerical ground the framework for correct interpretation of the experimental tomographic method. Thus using TDDFT as a theoretical basis, we have developed and used a highly parallel nonlinear nonperturbative algorithm and numerical scheme to explore the nonlinear response of molecule to ultrashort intense laser pulses. The main conclusion is that one must consider the increasing contribution of different orbitals to ionization with increasing laser intensity. This is a first step toward the development of a firm theoretical basis for the exploitation of high order harmonic generation in molecular orbital tomography with ultrashort intense laser pulses [2,5,15].

Acknowledgments. We thank P. B. Corkum and D.M. Villeneuve for *illuminating* discussions on orbital tomography. The authors would like to express their thanks to NSERC and CIPI for funding this research on the theory of the intense laser-molecule interaction and RQCHP, for parallel computer time. Finally, the authors are also thankful to Myriam Ziou, Mathieu Poulin and Catherine Lefebvre for the graphs and corrections done on this work.

References

1. Kienberger, R., et al.: Nature 427, 817 (2004)
2. Corkum, P.B., Krausz, F.: Nat. Phys. 3, 381 (2007)
3. Chu, X., Chu, S.: Phys. Rev. A 70, 061402 (2004); Dundas, D., Rost, J.M.: Phys. Rev. A 71, 013421 (2005)
4. Stapelfeldt, H., Seideman, T.: Rev. Mod. Phys. 75, 543 (2003)
5. Itatani, J., et al.: Nature (London) 432, 867 (2004)
6. Kamta, G.L., Bandrauk, A.D.: Phys. Rev. A 70, 011404 (2004); Kamta, G.L., Bandrauk, A.D.: Phys. Rev. A 71, 053407 (2005); Kamta, G.L., Bandrauk, A.D.: Laser Phys 15, 502 (2005)
7. Hohenberg, P., Kohn, W.: Phys. Rev. 136, B864 (1964)
8. Kohn, W., Sham, L.: Phys. Rev. 140, A1133 (1965)
9. Ullrich, C.A., Bandrauk, A.D.: Time-Dependent Density Functional Theory. Burke, K., Marques, M. (eds.), p. 357. Springer, New York (2006)

10. Casida, M.E.: Seminario, J.M. (ed.) Recent Developments and Applications in Modern Density-Functional Theory, p. 391. Elsevier, Amsterdam (1996); Furche, F.: *J. Chem. Phys.* 114, 5982 (2001)
11. Kümmel, S., Kronik, L.: *Rev. Mod. Phys.* 80, 3 (2008)
12. Zhang, Y., Yang, W.: *J. Chem. Phys.* 109, 2604 (1998); Dobbs, K.D., Dixon, D.A.: *J. Phys. Chem.* 98, 12584 (1994)
13. Garza, J., Nichols, J.A., Dixon, D.A.: *J. Chem. Phys.* 112, 7880 (2000)
14. Penka, E.F., Bandrauk, A.D.: *Can. J. Chem.* (accepted, 2009)
15. Pavičić, D., et al.: *Phys. Rev. Lett.* 98, 243001 (2007)
16. Alnaser, A.S., et al.: *Phys. Rev. A* 71, 031403 (2005)
17. Kohn, W., Sham, L.J.: *Phys. Rev.* 140, A1133 (1965)
18. Runge, E., Gross, E.K.U.: *Phys. Rev. Lett.* 52, 997 (1984)
19. Nguyen, H.S., Bandrauk, A.D., Ullrich, C.A.: *Phys. Rev. A* 69, 063415 (2004)
20. Koopmans, T.: *Physica* 1, 104 (1933); Mulliken, R.S.: *Phys. Rev.* 74, 736 (1948)
21. Szabo, A., Ostlund, N.S.: Introduction to Advanced Electronic Structure Theory (Paperback 1996)
22. Janak, J.F.: *Phys. Rev. B* 18, 7165 (1978)
23. Lee, C., Yang, W., Parr, R.G.: *Phys. Rev. B* 37, 785 (1988)
24. Becke, A.D.: *J. Chem. Phys.* 104, 1040 (1996); Becke, A.D.: *J. Chem. Phys.* 107, 8554 (1997)
25. Leeuwen, R.V., Baerends, E.J.: *Phys. Rev. A* 49, 2421 (1994)
26. Umezawa, N.: *Phys. Rev. A* 74, 032505 (2006)
27. Gisbergen, S.J.A.v., Snijders, J.G., Baerends, E.J.: *J. Chem. Phys. Lett.* 109 (1998); Banerjee, A., Harbola, M.K.: *Phys. Rev. A* 60, 3599 (1999); Stener, M., Decleva, P.: *J. Chem. Phys.* 112, 10871 (2000)
28. Perdew, J.P., Zunger, A.: *Phys. Rev. B* 23, 5048 (1981)
29. Sharp, R.T., Horton, G.K.: *Phys. Rev.* 90, 317 (1953); Talman, J.D., Shadwick, W.F.: *Phys. Rev. A* 14, 36 (1976)
30. Krieger, J.B., Li, Y., Iafrate, G.J.: *Phys. Rev. A* 45, 101 (1992); Krieger, J.B., Li, Y., Iafrate, G.J.: *Phys. Rev. A* 46, 5453 (1992)
31. Krieger, J.B., Li, Y., Iafrate, G.J.: *Phys. Rev. A* 47, 165 (1993)
32. Grabo, T., Gross, E.K.U.: *Chem. Phys. Lett.* 240, 141 (1995); Grabo, T., Gross, E.K.U.: *Int. J. Quantum Chem.* 64, 95 (1997); Stowasser, R., Hoffmann, R.: *J. Am. Chem. Soc.* 121, 3414 (1999)
33. Troullier, N., Martins, J.L.: *Solid State Commun.* 74, 13 (1990); Troullier, N., Martins, J.L.: *Phys. Rev. B* 43, 1993 (1991); Castro, A., et al.: *Phys. Stat. Sol. B* 243, 2465 (2006)
34. in Université de Sherbrooke (Sherbrooke), <https://rqchp.ca/page/0/FR>
35. Brabec, T., Krausz, F.: *Rev. Mod. Phys.* 72, 545 (2000)
36. Crank, J., Nicolson, P.: *Proc. Camb. Phil. Soc.* 43, 50 (1947)
37. Fevens, T., Jiang, H.: *SIAM J. Sci. Comput.* 21, 255 (1999); Bandrauk, A.D., Lu, H.Z.: *Phys. Rev. A* 72, 023408 (2005)
38. Jiang, H., Novak, I.: *J. Mol. Struct.* 645(2-3), 177 (2003); Hubert, K., Herzberg, G.: Molecular Spectra and Molecular Structure. Van Nostrand Reinhold, New York (1979)
39. Lias, S.G.: In Ionization Energy Evaluation in NIST Chemistry WebBook, NIST Standard Reference Database. Linstrom, W.G.M.P.J. (ed.) National Institute of Standards and Technology, <http://webbook.nist.gov>, Gaithersburg MD 20899, vol. 69 (March 2003); Kikoin, I.K.: Tables of Physical Quantities (in Russian). Atomizdat, Moscow (1976)
40. Vosko, S.H., Wilk, L., Nusair, M.: *Canadian Journal of Physics* 58, 1200 (1980)
41. Perdew, J.P., et al.: *Physical Review B* 46, 6671 (1992)
42. Cohen, A.J., Handy, N.C.: *Molecular Physics* 99, 607 (2001)

Considerations for Progressive Damage in Fiber-Reinforced Composite Materials Subject to Fatigue

John Montesano, Kamran Behdinan, Zouheir Fawaz, and Cheung Poon

Department of Aerospace Engineering, Ryerson University, 350 Victoria St.,
Toronto, ON, Canada, M5B 2K3
{gmontesa,kbehdinazfawaz,c1poon}@ryerson.ca

Abstract. Due to the increased use of composite materials in the aerospace industry, numerous attempts have been made to develop fatigue models in order to predict the fatigue behaviour and consequently the fatigue life of these materials. Existing fatigue models have significant deficiencies, thus are not widely acceptable in the industry. A better understanding of the exhibited fatigue behaviour of composite materials is consequently required. The complex nature of fatigue behaviour in fiber-reinforced composite materials is presently investigated. An explicit progressive damage model, that is mechanistic in nature, is currently being developed using the concept of a representative volume element. A micromechanical finite element model that is capable of explicit damage initiation and propagation modeling is utilized for simulation of damage development. The predicted numerical results illustrate the capabilities of the current model. Future work is also outlined in the paper as the development of the fatigue model is continued.

Keywords: Composite materials, fiber-reinforced / epoxy laminates, fatigue, progressive damage modeling, micromechanical.

1 Introduction

The use of composite materials in many industries including the aerospace industry has considerably increased in recent years. Due to their light-weight and high-strength, composite materials are becoming the preferred choice over more conventional metallic alloys for the design of aircraft structures and components. The obvious benefit in using composite materials for aircraft structures is the significant decrease in fuel consumption due to the reduction of aircraft structural weight. A better understanding of composite material behaviour is consequently warranted for further use of these materials. More specifically, the nature of composite material behaviour when subjected to cyclic or fatigue loading is of primary interest for predicting the lifetime or fatigue life of a particular composite component. Fatigue loading is accountable for a high percentage of service failures in the aerospace industry, thus understanding the influence of fatigue loading for components is critical.

Composites consist of two or more distinct constituent materials or phases at the macroscopic level, which are combined to form an inhomogeneous and anisotropic material. Continuous unidirectional fiber-reinforced laminates are a specific type of

composite material commonly used in the aircraft industry. The laminate consists of two or more lamina or plies that are layered in a particular stacking sequence with various orientations relative to one another [1]. Each ply consists of the two constituent materials, i.e., the reinforcing fibers and the binding matrix. A schematic of a typical laminate lay-up and of a characteristic ply constituency is shown in Fig. 1. Laminates are markedly different than metallic alloys due the inhomogeneous nature of the material, which creates a difficulty when determining its fatigue properties. Components made from metallic alloys typically exhibit cracking due to fatigue loading at locations of high stress concentrations. Laminate components on the other hand can exhibit damage throughout the structure without any explicit stress concentrations locations. Laminates typically exhibit many distinct mechanisms of damage or failure during fatigue loading. The prominent damage mechanisms include matrix cracking, fiber fracture, fiber-matrix interface debonding and delamination between adjacent plies [2]. Furthermore, the interaction of these damage mechanisms has been experimentally observed to have an influence on the fatigue behaviour of laminates [1], [2]. In a homogeneous material, such as a metallic alloy, once a dominant crack is formed after a number of load cycles, the fatigue life can be determined if the initial crack size and the growth behaviour of the crack are known. For metallic alloys, simple linearly elastic fracture mechanics models are often used to simulate fatigue behaviour since the material properties (such as stiffness or strength) are unaffected or only slightly affected by fatigue loading. This is not the case for composite materials since the aforementioned damage mechanisms are rather diffuse throughout the material. Also since damage commences after only a few load cycles, there is gradual stiffness loss in the damaged areas of the material which leads to a continuous redistribution of stress during cyclic loading. The accurate prediction of a composite components fatigue behaviour, and thus its fatigue life, is consequently a difficult task. Generalized methods for the prediction of composite material fatigue life have not been adopted in the industry. As a result, many researchers have developed fatigue models in an attempt to predict the fatigue life or fatigue behaviour of various composite components with the purpose of better understanding these materials.

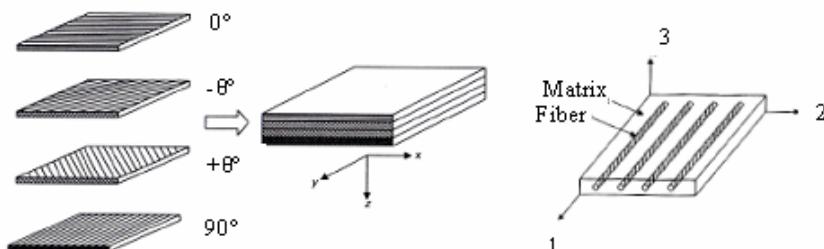


Fig. 1. Schematic of typical 4-ply laminate and ply constituency [1]

2 Literature Review

There has been great effort by researchers to develop methods for modeling fatigue damage and to predict the fatigue life of composites. Existing fatigue models for

fiber-reinforced polymeric composites are empirical, semi-empirical, phenomenological, statistical, or mechanistic in nature. These models can generally be classified into three main categories: empirical/semi-empirical models based on experimental stress-life (S-N) curves and not actual damage mechanisms or damage accumulation, phenomenological residual life/strength/stiffness models based on damage accumulation, and mechanistic progressive damage models which are based on specific damage mechanisms.

2.1 Empirical/Semi-empirical Models

Empirical/semi-empirical models do not account for damage accumulation or specific damage mechanisms such as matrix cracking. Rather, these models quantify failure or determine the composite fatigue life based solely on a fixed loading condition (i.e., the stress state). In either case, these models require corresponding experimental testing data in the form of S-N curves. Hashin and Rotem [3] developed a fatigue failure criterion for fiber reinforced composites where the fiber and matrix constituents each have a separate failure criterion. The failure criterion is based on a fixed stress state in the material, and the assumption that failure of one constituent means failure of the material. Also, the model is only valid for laminates with unidirectional plies. Reifsnider and Gao [4] proposed a similar model based on an average stress formulation of unidirectional composites, which accounted for the constituent material properties. Ellyin and El-Kadi [5] developed a fatigue criterion for unidirectional laminates which related the total energy input due to loading and the fatigue life of the material. The prediction of fatigue life is based on the fixed stress state and the material constants which are found experimentally. Fawaz and Ellyin [6] proposed a linear relationship between the applied cyclic stress and the number of cycles for unidirectional laminates. Fatigue life can be determined from the applied cyclic stress and experimental material constants. Bond [7] also developed a semi-empirical model to predict fatigue life of unidirectional laminates based on the applied cyclic stress. These experimentally based models are all specific to certain types of composite materials, and do not consider specific damage mechanisms in their formulation which is seen as a drawback.

2.2 Residual Life/Stiffness/Strength Models

Residual life/stiffness/strength models are phenomenological in nature and are based on the concept of cumulative damage. These models do not account for specific damage mechanisms, but rather treat damage accumulation on the macroscopic scale. One of the first residual life cumulative damage models was developed by Miner [8] for metals, which characterized damage (D) as a function of life fraction (i.e., $D = n/N$, where n is the load cycle and N is the fatigue life for a given stress state). In order to determine the cumulative damage, the fatigue life of the material must be known for a given loading stress from experimental testing. Other similar cumulative damage models were developed for composites, based on the use of life fraction for single stage and multi-stage fatigue loading scenarios [9], [10], [11], [12]. The obvious deficiency with these primitive fatigue models is the lack of any physically-based criteria for predicting damage, not accounting for the fiber and matrix material properties in the formulations, and not

considering the effect of loading history on damage accumulation. Also, these analytical models are all specific to a certain type of composite material and cannot be used in a general case for prediction of damage accumulation.

The residual stiffness models use the degradation of the material stiffness to determine the fatigue life of a composite. Hahn and Kim [13] and O'Brien and Reifsnider [14] each proposed a secant modulus approach, where fatigue failure is assumed to occur when the secant modulus reduces to a critical value. It was shown that damage growth and stiffness loss were load history dependent, thus the secant criterion is not valid for general applications. One of the early residual stiffness models was proposed by Hwang and Han [12], [15]. The material stiffness decreases with every increasing load cycle, and failure occurs when the damage parameter (D), which is based on the changing stiffness value, is unity. Similar residual stiffness models were proposed by Whitworth [16], [17] and Yang et al [18], [19] for unidirectional composites. Hansen [20] and VanPaepegem and Degrieck [21] employed similar stiffness degradation models for woven composites and implemented them into finite element analysis models.

The residual strength models use the degradation of the material residual strength to determine the fatigue life of a composite. Many of these models are incorporated within statistical models to predict the probability of fatigue failure. One of the early residual strength models was developed by Broutman and Sahu [22] for unidirectional composites. They found that the residual strength is a linear function of the number of cycles spent at a particular stress level. Hahn and Kim [23] also developed a linear model relating residual strength and number of load cycles. The model predicts failure when the residual strength reduces to the level of the maximum applied load. This is characteristic of many residual strength models developed thereafter [24], [25], [26], [27], [28].

2.3 Progressive Damage Models

Progressive damage models are mechanistic in that they consider specific micromechanical damage mechanisms such as matrix cracking, fiber-matrix debonding, delamination and fiber fracture in order to simulate the development of damage. These models are based on a physical interpretation of the actual damage mechanisms, where the damage criterion is based on micromechanical formulations which are often used in the context of finite element (FE) analysis. These models can be classified into three subcategories: models that predict damage growth, models that correlate damage growth with material property degradation, and models that explicitly model the damage mechanisms.

Models that predict actual damage growth consider one specific damage mechanism, and determine the physical change in that damage with increasing loading cycles. These models are typically of the form of the well-known Paris Law for crack propagation in homogeneous materials (i.e., da/dN). Notched laminates are typically employed to initiate the specific damage type required. One such model is that developed by Bergmann and Prinz [29], where delamination area growth is predicted for unidirectional laminates. Feng et al [30] developed a model for predicting matrix crack growth in unidirectional laminates, using maximum strain-energy release rate along with experimental constants in the formulation. A similar model was proposed

by Henaff-Gardin et al [31], where the increase in area of damage due to matrix cracking was simulated. Schon [32] proposed a model for delamination growth in unidirectional composites, where growth is also based on energy release rate.

Models that correlate damage growth with material property degradation typically employ FE models to simulate damage progression and to predict fatigue life of a component. Some of the early researchers proposed models to represent stiffness reduction of unidirectional laminates due to matrix cracking and/or delamination. One of the first models to correlate matrix cracking with stiffness loss was the shear-lag model developed by Highsmith and Reifsnider [33]. The quantity of matrix cracking was experimentally observed and correlated to the specimen stiffness. These observations were then implemented into analytical differential equations to predict stiffness loss. The model does assume that cracks cannot interact with one another, which is not physically accurate. Hashin [34] later proposed a variational model that considers matrix cracking to predict stiffness degradation. The results presented are an improvement when compared to previous work [33]. Other similar models correlating matrix cracking and stiffness loss were subsequently developed by El-Mahi et al [35], Smith and Ogin [36] and Katerelos et al [37]. Moreover, Reifsnider [38] proposed a critical element method to correlate stiffness and residual strength degradation with damage mechanisms in unidirectional composites. The model assumes the material properties of the sub-critical elements degrade based on specific damage mechanisms (i.e., matrix cracking), which subsequently degrades the residual strength of the critical elements. Sendeckyj [39] found that the critical element method prediction is inconsistent with experimental observations. Other models have been developed which are based on a continuum mechanics foundation, specifically termed as continuum damage mechanics (CDM) based fatigue models. One of the first CDM based model for unidirectional composites was developed by Laws et al [40]. A scalar damage parameter is formulated based on matrix crack density and implemented into the composite material constitutive equations to reduce the material stiffness. Talreja [41] also proposed a damage vector based on matrix cracking, which was implemented into the constitutive equations to degrade the material stiffness in the context of CDM. This model was later extended to include delamination damage [42]. Allen et al [43], [44] and Coats and Harris [45] developed a CDM based model for laminates based on internal state variables which represented matrix cracking in the form of a second-order damage tensor. This tensor is used in the material constitutive equations to degrade stiffness. Spearing et al [46] considered matrix cracking and delamination in unidirectional laminates, employing a FE model to simulate stiffness degradation; note that a simplified analytical model is used for stiffness degradation. Ladeveze and LeDantec [47] developed a model for laminates in which matrix cracking and fiber-matrix debonding are considered to formulate damage variables. These variables are used to degrade the material stiffness. Shokrieh and Lessard [48], [49] developed a generalized residual material property degradation model for fatigue predictions of unidirectional laminates with arbitrary geometry and stacking sequence. Failure criterion is based on specific damage modes, where material properties are consequently degraded within the context of a three-dimensional FE model. Camanho and Matthews [50] proposed a similar numerical model for prediction of damage progression at fastener holes locations in composite laminates.

Models that explicitly represent damage mechanisms within the context of FE analysis typically employ a number of varying numerical techniques to simulate matrix cracking, delamination, fiber-matrix debonding, etc. Due to the micro-scale of typical composite constituent materials, it is impossible to model an entire composite specimen in full even with the high performance computational platforms available today. This motivated Sun and Vaidya [51] to develop the concept of a representative volume element (RVE) or repeated unit cell (RUC), which enables simulation of a composite material by an equivalent fiber-matrix representation. This allows for a drastic reduction in computational effort. In order to allow for a mechanically sound physical representation of the composite material, the RVE or RUC must be adequate to capture all essential failure or damage modes. Xia et al [52] developed a micromechanical FE model for unidirectional laminates using the RVE approach. Matrix cracking is considered in the model, which employs an equivalent strain failure criterion to explicitly model the matrix cracks. Many other explicit damage models, with similar failure criteria, also utilizing a RVE were proposed for both unidirectional and braided composites [53], [54], [55]. The main drawback with the aforementioned models is that they do not attempt to simulate fatigue loading, which is an obvious deficiency. Lubineau et al [56] developed a so-called micro/meso-mechanical model, which can be considered a hybrid damage model. Some damage is represented in the context of CDM (i.e., meso-scale for fiber-matrix debonding) or by explicit modeling (i.e., micro-scale for delamination and matrix cracking) within the same damage model. This model was later used for a thermal fatigue loading simulation of unidirectional laminates [57]. The damage paths were predetermined in the FE model, which is seen as the main deficiency. Shi and Zhang [58] developed a micromechanical FE model in which analytical damage models were used to simulate matrix crack and fiber-matrix debonding interaction. The model geometry is oversimplified, but tension-tension fatigue simulations are accomplished on a cycle-by-cycle basis.

2.4 Overview

It is in the opinion of the authors that progressive damage models are superior to empirical/semi-empirical or phenomenological models to predict fatigue behaviour in composites. The mechanistic nature of progressive damage models allows for a more accurate physical interpretation of the innate complexities of the damage mechanisms and their interactions. This is vital for accurately predicting fatigue properties and thus fatigue life. More specifically, progressive damage models based on explicit damage modeling are regarded as the most accurate prediction models. They allow for a more ‘natural’ progression of damage, versus employing empirical factors which tend to ‘guide’ damage progression. The main drawback of this approach is the increased complexity of the fatigue model, and the increased demand for computational effort (due to refined FE mesh, specifically if cycle-by-cycle simulations are performed). With emerging computational technologies, applications of explicit progressive damage models will undoubtedly increase. The use of advanced parallel computing platforms, such as those offered by HPCVL, will enable these types of fatigue models to be used in a practical sense.

The objective of this research is to develop an explicit micromechanical progressive damage model for composite material fatigue simulations. Specifically, the focus

is to study the progression of damage mechanisms in order to develop an increased comprehension of composite material fatigue behaviour. The remaining sections in this paper will focus on presenting the current simulation model with preliminary prediction results, and outlining the remaining work that is required in the ongoing development of the model.

3 Numerical Model

The progressive damage model is currently in the developmental stage, and it must be clearly stated that this is work in progress. As mentioned, an explicit mechanistic progressive damage model is developed within the framework of a RVE approach. Thus, the fibers and matrix constituents are included in the model. For this study, a unidirectional glass fiber-reinforced epoxy lamina (single ply) is regarded in the simulations. This type of composite is considered since experimental testing of a similar material will be conducted in a future study.

3.1 Constituent Material Properties

The lamina consists of R-glass fibers and a thermoset epoxy matrix. Typical values of elastic modulus, Poisson's ratio and ultimate tensile strength for these constituents are listed in Table 1. The volume fraction for the lamina is assumed to be 0.50 for the fibers and matrix. Note at this stage in model development, both the fibers and matrix are assumed to exhibit linearly elastic material behaviour. Although this assumption may be true for the glass fibers, the epoxy matrix can exhibit nonlinear behaviour during high-strain or elevated-temperature loading. At this stage in the model development, this assumption is made in order to simplify the solution.

Table 1. Constituent material properties [59]

Constituent Material	Modulus (MPa)	Poisson's Ratio	Ult. Strength (MPa)
R-glass fiber	86000	0.20	3200
Epoxy matrix	4500	0.40	130

3.2 Finite Element Model

The FE model is developed within the framework of a RVE approach, thus the RVE must be defined. It has been found that when the in-plane dimensions of a composite are much larger than the dimensions in the thickness direction, the stress and strain fields in the composite will be periodic in nature (represented by a RVE) [52]. Consider the rectangular lamina specimen subjected to an axial tensile load as shown in Fig. 2. With the assumption that the fibers are evenly distributed in a uniform array within the lamina, symmetry can be utilized as shown. The RVE can then be formed at the specified location. The RVE employed in this study is shown in Fig. 3. This arrangement of fibers was previously shown to adequately represent the interaction of damage mechanisms between fibers in a unidirectional lamina [60], and thus is used in this study. In reality, a unidirectional composite will consist of randomly oriented

fibers within the epoxy matrix. At this stage in model development, this idealization of the geometry is deemed sufficient. The size of the square cross-section of the RVE is 125 μm , while the depth is 40 μm . The diameter of the round fibers is 35 μm (typical for glass fibers used in composites), which ensures the volume fraction within the RVE is maintained at 0.50 for both the fibers and matrix.

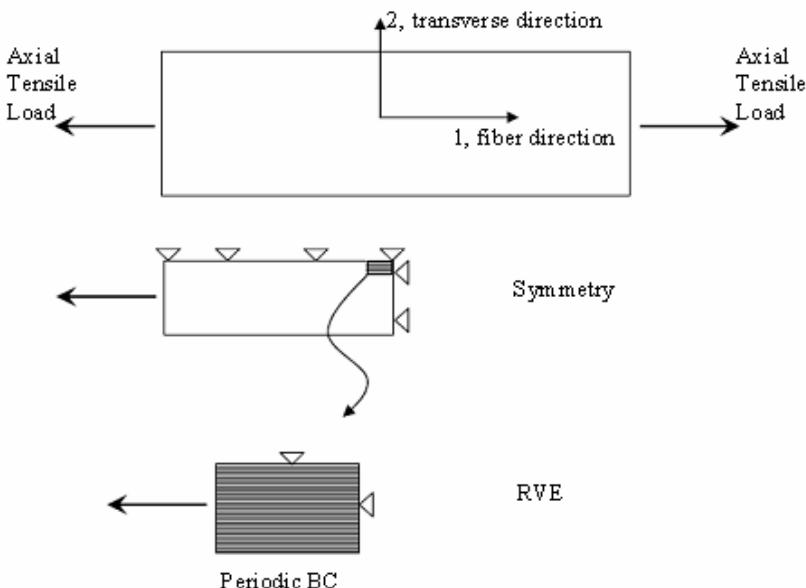


Fig. 2. RVE schematic

The commercial FE software package ABAQUS is employed for analysis. A typical FE mesh for the constituent based RVE is also shown in Fig. 3. Note that since fiber-matrix debonding is not yet considered, the interface of the fibers and matrix is assumed to be fixed together without any relative motion. The elements chosen for the analysis are 3D 10-node tetrahedral elements; the mesh shown contains 48,520 elements. Care must be taken in order to minimize the size of the elements to ensure continuity of the damage progression (i.e., crack propagation as discussed in Section 3.3). Preliminary testing of the FE model with this degree of mesh refinement has proven to be adequate for damage progression in the presented composite. The RVE boundary conditions (BC's) are chosen to ensure kinematic compatibility at the model faces. Due to three planes of symmetry, the three indicated faces in Fig. 3 have symmetry BC's (i.e., restricted motion normal to the face). Also since the RVE is assumed to be repeated throughout the composite lamina, two faces are defined with periodic BC's (i.e., the face must remain plane during deformation). This method of applying periodic BC to RVE's has been successfully employed in the literature [52]. The loaded face is stipulated to have a specified normal displacement, which ensures that the loaded face remains flat. This simulates a displacement controlled uniaxial static test load.

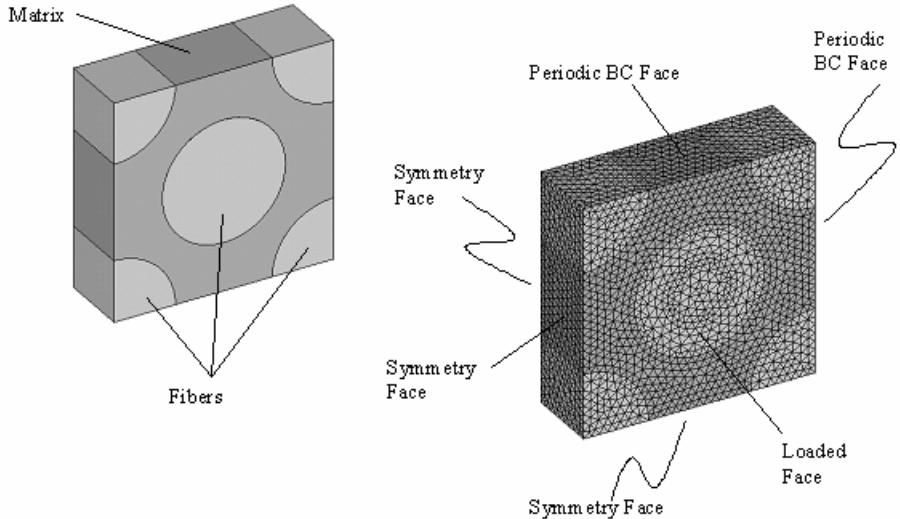


Fig. 3. RVE representation, mesh and boundary conditions

3.3 Damage Model

As mentioned at this stage in development of the model, fiber-matrix debonding and delamination are not considered. Cracking of the matrix and fiber are however included in the current model. Although the loading on the RVE is uniaxial, the elements in the FE mesh will exhibit a tri-axial state of stress. A multi-axial damage criterion is consequently required. The model uses maximum equivalent strain theory as the failure criteria for the fiber and matrix elements in the FE mesh. This ensures a natural initiation and propagation of cracking without empirical factors in the model. The equivalent strain for each element is determined from the principal strains in the three principal directions (i.e., ε_1 , ε_2 , ε_3). The corresponding equation for equivalent or Von Mises strain is given by:

$$\varepsilon_{eq} = \frac{1}{1+\nu} \sqrt{\frac{1}{2} \left[(\varepsilon_1 - \varepsilon_2)^2 + (\varepsilon_2 - \varepsilon_3)^2 + (\varepsilon_3 - \varepsilon_1)^2 \right]} \quad (1)$$

Once the element equivalent strain reaches a critical value, the element is assumed to have failed. The corresponding limit expression for the matrix strain is given by:

$$\varepsilon_{eq}^m \geq \varepsilon_{cr}^m \quad (2)$$

A similar expression can be defined for the fiber strain as:

$$\varepsilon_{eq}^f \geq \varepsilon_{cr}^f \quad (3)$$

The critical strain (ε_{cr}) of the matrix and fiber are thresholds below which no damage occurs. These values are assumed to be constants, and are determined from the ultimate

tensile stresses. With the aforementioned linearly elastic material assumptions, the critical strains can be found using Hooke's law. From Table 1, the critical strains for the matrix and fibers are found to be 0.0289 and 0.03721 respectively.

Once an element fails, it is no longer effective in carrying any stress under the applied tensile load. This simulates the void caused by a crack initiating or propagating into a region of the composite material. This explicit representation of cracking is modeled via a common element-deletion method, which has been employed by many researchers [52]. When an element fails it is not actually removed from the FE mesh, but the stiffness of that element is reduced to a near-zero value (note that making the stiffness of an element exactly zero will cause numerical instability, which will result in computational convergence difficulties). Thus, the element stiffness matrix $[K]$ is scaled by the factor β . The corresponding effective element stiffness matrix is given by:

$$[K]^{eff} = \beta[K] \quad (4)$$

If an element is fully effective $\beta = 1$, but when an element has failed β is set to 10^{-6} which is found to be sufficient in this study.

3.4 Simulation Procedure

At this stage in the development of the model, a quasi-static tensile simulation is conducted in lieu of a fatigue simulation. Although the model is intended for fatigue simulations, the quasi-static simulations (consisting of increasing the applied load in an incremental fashion) will illustrate the predictive capabilities of the current model to simulate progressive damage. The parametric capabilities of ABAQUS script files are used to conduct the numerical simulations. The script files (or input files) are created using Python, an object oriented algorithm which is built-in to the ABAQUS environment.

The first step in the simulation is to input the geometry, and define both the mesh and boundary conditions. As indicated the geometry of the RVE is simplified, thus the automated geometry algorithm is primitive at this stage in model development. Guided meshing parameters are also employed within the ABAQUS script file to generate the mesh of the RVE. Next, the initial solution of the quasi-static analysis is conducted, and the matrix and fiber elements are checked for failure using the equivalent strain criteria defined by Eqs. (2) and (3). The corresponding failed elements are 'deleted', the load is increased, and the process is repeated for a predetermined number of iterations. A schematic of the simulation is shown in Fig. 4. Note that the damaged elements are tracked every load iteration. A corresponding output file is created in order to allow for post-processing of the damage data, where the formation of graphical images of the damage progression is feasible.

4 Simulation Results

The FE mesh shown in Fig. 3 is used for the simulation, which was found to be adequately refined. An initial displacement of 1.1 μm is applied at the loaded face. Sun

and Vaidya [51] have shown that the average axial strain along the loading direction in the presented RVE can be found by:

$$\varepsilon_{11}^{avg} = \frac{u_{load}}{L} \quad (5)$$

The dimension of the RVE in the loading direction is L , where u_{load} is the applied displacement on the loaded face of the RVE. The specified initial displacement of 1.1

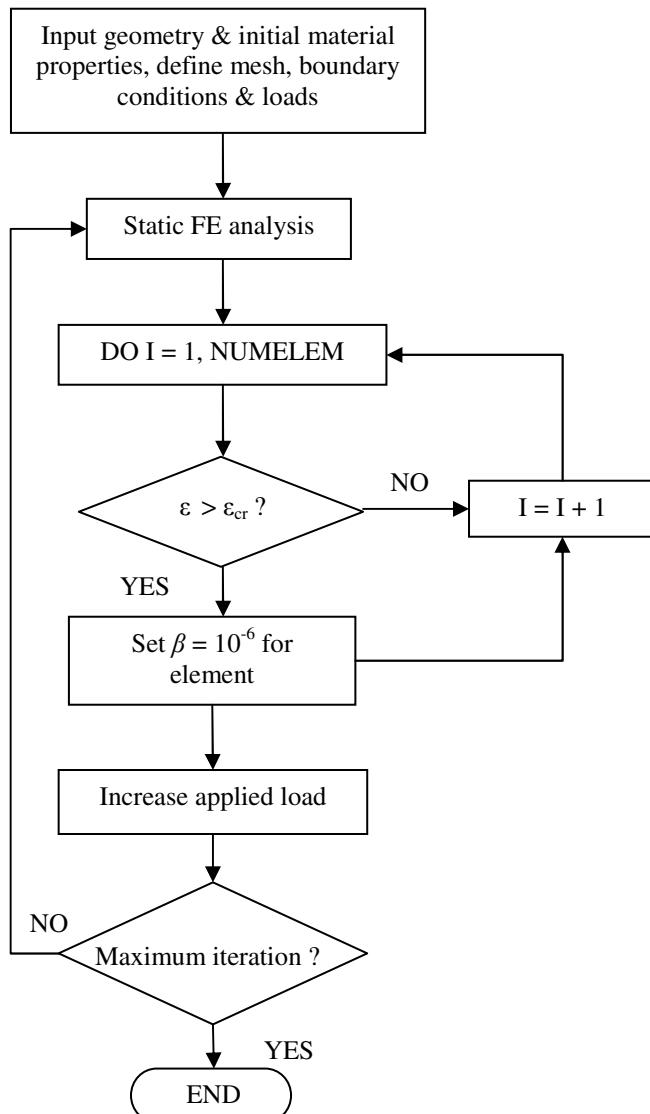


Fig. 4. Simulation flowchart

μm with $L = 40 \mu\text{m}$ yields an average axial strain of 0.0275. This strain is slightly below the critical strain of the matrix found from Table 1, which is a good starting point for the quasi-static simulation.

The progression of cracking for the quasi-static simulation is illustrated in Fig. 5 for some of the simulation iterations. The images in Fig. 5 were created using the output damage data file and the graphical capabilities of ABAQUS. Note that damaged or ‘deleted’ elements are indicated in white. As shown, the damage is primarily in the matrix material which is not surprising since the critical strain for the matrix is lower than that of the fiber (i.e., 0.0289 versus 0.03721, respectively). After only a few iterations, matrix cracks initiate near the fibers as seen in Fig. 5(a). This is commonly observed in unidirectional laminates, and is caused by the increased local stress at the fiber-matrix interface due to the varying elastic behaviour of the fibers and the matrix. Damage continues to progress as the applied load is increased, and additional matrix cracks initiate as shown in Fig. 5(b). The direction of matrix crack propagation in unidirectional laminates is typically normal to the loading direction, which is also illustrated in Fig. 5. Towards the later half of the simulation, individual matrix cracks begin to coalesce, and damage in the matrix is more widespread (Fig. 5(c) and (d)).

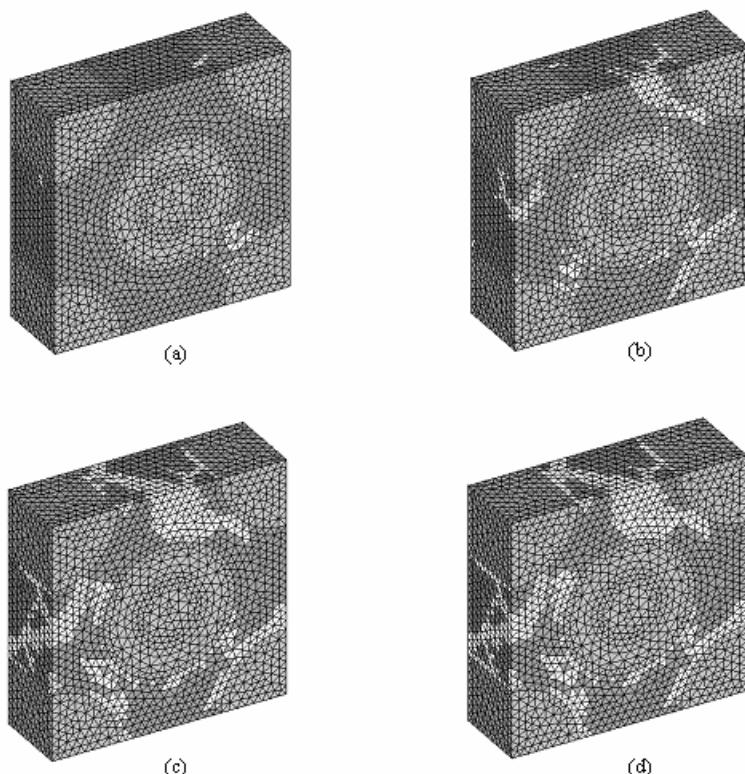


Fig. 5. Damage progression: (a) matrix crack initiation near fiber interface, (b) crack progression after a few iterations, (c) further crack development, (d) cracking is widespread between the adjacent fibers after the final iteration.

The simulation is terminated before complete failure of the RVE. The predicted damage progression is comparable to similar predictions found in the literature, where matrix cracking tends to be dominant near the fiber-matrix interface and around the perimeter of the fibers [55], [61]. This illustrates a key point in damage progression of composites; mainly that matrix cracking can influence both fiber-matrix debonding and fiber fracture. Again note that fiber-matrix debonding is not included in the current simulation model. Although the model is in the developmental stages, these preliminary results are certainly promising.

5 Discussion and Conclusion

The current simulation model provides a means to predict explicit damage progression within the context of a RVE. Although the current simulation model is a long way from completion, the predicted quasi-static simulation results are encouraging at this stage. The long-term objective of developing an explicit mechanistic progressive damage model for fatigue simulations of composite materials is indeed promising.

The presented simulation, which consisted of 50 load iterations and a linear FE model consisting of a 48,520 element mesh, required an average of 60 minutes to complete on a PC platform with a Pentium Duo Core 2.4 GHz processor. As mentioned, the simulation was terminated before complete failure of the RVE. For an explicit progressive damage model using a RVE, failure of the RVE must imply failure of the specimen under investigation. For this reason, a larger RVE may be required for a particular analysis which will undoubtedly increase computation time. Also, the addition of nonlinear material behaviour and further damage mechanisms will require further computation time. Furthermore, a full fatigue simulation, which may be based on a cycle-by-cycle analysis, will require an increased number of solution iterations. It is not practical to execute such a simulation on a simple PC platform. The need for high performance computing resources, such as the parallel processing platforms offered by HPCVL, becomes apparent in order to perform the required simulations. This will enable the current simulation model to be utilized for fatigue behaviour analysis, and eventually for fatigue life prediction. As mentioned, this type of fatigue model is necessary for accurate representation of the composite material and for accurate simulation of fatigue damage progression.

The model is currently under development with the intentions of further utilizing HPCVL resources. Further work includes improvement of the material model for the epoxy matrix. As indicated the current simulation model utilizes a linear elastic material model for the matrix, which will in fact require a nonlinear viscoelastic model. The long term goal of this project is to conduct fatigue simulations at elevated temperatures, thus a nonlinear viscoelastic material model is necessary. Also, the addition of fiber-matrix interface debonding is a necessary damage mechanism. This can be done via interface elements within the FE model, or through an analytical damage model. Additional work includes the execution of a fatigue simulation using the current FE model. Two methods are currently under investigation: the first is a simulation based on cycle-by-cycle analysis, while the second is based on an analytical approach whereby the influence of the number of fatigue cycles is considered to model damage progression. Furthermore, the addition of a statistical algorithm to consider the random arrangement of fibers in the

matrix will be added to the simulation model. An algorithm to automatically generate RVE's for unidirectional laminates and other types of composites such as woven or braided fiber-reinforced composites is currently under development. Finally, the numerical model will be validated with corresponding experimental fatigue test data conducted at elevated temperatures.

Upon completion, the simulation model will provide a means to predict damage progression in composite materials during fatigue. This will provide insight on the various fatigue damage mechanisms and their interactions, which will ultimately provide an increased understanding of fatigue behaviour in composite materials. Note that the intention of the simulation model is not to create a tool to predict the fatigue life of composites. Note however that based on the findings of the study, a practical predictive tool for fatigue life may be consequently developed.

Acknowledgements. The authors would like to thank the High Performance Computing Virtual Laboratory (HPCVL) in Canada for funding in the form of a scholarship to the first author, and for computational resources in support of the numerical portion of this research.

References

1. Agarwal, B.D., Broutman, L.J., Chandrashekara, K.: Analysis and Performance of Fiber Composites. John Wiley and Sons Inc., Hoboken (2006)
2. Harris, B.: A Historical Review of the Fatigue Behaviour of Fibre-Reinforced Plastics. In: Harris, B. (ed.) Fatigue in Composites, pp. 3–35. Woodhead Publishing Ltd., Cambridge (2003)
3. Hashin, Z., Rotem, A.: A Fatigue Criterion for Fibre Reinforced Composite Materials. *J. Comp. Mater.* 7, 448–464 (1973)
4. Reifsneider, K.L., Gao, Z.: A Micromechanics Model for Composites Under Fatigue Loading. *Int. J. Fatigue* 13, 149–156 (1991)
5. Ellyin, F., El-Kadi, H.: A Fatigue Failure Criterion for Fiber Reinforced Composite Laminae. *Comp. Struct.* 15, 61–74 (1990)
6. Fawaz, Z., Ellyin, F.: Fatigue Failure Model for Fibre-Reinforced Materials Under General Loading Conditions. *J. Comp. Mater.* 28, 1432–1451 (1994)
7. Bond, I.P.: Fatigue Life Prediction for GRP Subjected to Variable Amplitude Loading. *Comp. Part A* 30, 961–970 (1999)
8. Miner, M.A.: Cumulative Damage in Fatigue. *J. App. Mech. (ASME)* 12, A159–A164 (1945)
9. Marco, S.M., Starkey, W.L.: A Concept of Fatigue Damage. *Trans. ASME* 76, 627–632 (1954)
10. Howe, R.J., Owen, M.J.: Accumulation of Damage in a Glass-Reinforced Plastic Under Tensile and Fatigue Loading. In: Proceedings of the 8th International Reinforced Plastics Congress, pp. 137–148. British Plastics Federation, London (1972)
11. Hashin, Z., Rotem, A.: A Cumulative Damage Theory of Fatigue Failure. *Mat. Sci. Eng.* 34, 147–160 (1978)
12. Hwang, W., Han, K.S.: Cumulative Damage Models and Multi-stress Fatigue Life Prediction. *J. Comp. Mater.* 20, 125–153 (1986)

13. Hahn, H.T., Kim, R.Y.: Fatigue Behavior of Composite Laminates. *J. Comp. Mater.* 10, 156–180 (1976)
14. O'Brien, T.K., Reifsnider, K.L.: Fatigue Damage Evaluation Through Stiffness Measurements in Boron-Epoxy Laminates. *J. Comp. Mater.* 15, 55–77 (1981)
15. Hwang, W., Han, K.S.: Fatigue of Composites - Fatigue Modulus Concept and Life Prediction. *J. Comp. Mater.* 20, 154–165 (1986)
16. Whitworth, H.A.: Modeling Stiffness Reduction of Graphite/Epoxy Composite Laminates. *J. Comp. Mater.* 21, 362–372 (1987)
17. Whitworth, H.A.: Cumulative Damage in Composites. *J. Eng. Mater. Technol.* 112, 358–361 (1990)
18. Yang, J.N., Jones, D.L., Yang, S.H., Meskini, A.: A Stiffness Degradation Model for Graphite/Epoxy Laminates. *J. Comp. Mater.* 24, 753–769 (1990)
19. Yang, J.N., Lee, L.J., Sheu, D.Y.: Modulus Reduction and Fatigue Damage of Matrix Dominated Composite Laminates. *Comp. Struct.* 21, 91–100 (1992)
20. Hansen, U.: Damage Development in Woven Fabric Composites During Tension-Tension Fatigue. *J. Comp. Mater.* 33, 614–639 (1999)
21. VanPaepegem, W., Degrieck, J.: Experimental Set-up for and Numerical Modelling of Bending Fatigue Experiments on Plain Woven Glass/Epoxy Composites. *Comp. Struct.* 51, 1–8 (2001)
22. Broutman, L.J., Sahu, S.: A New Theory to Predict Cumulative Fatigue Damage in Fibre-glass Reinforced Plastics. In: *Composite Materials: Testing and Design*, pp. 170–188. ASTM STP 497, Philadelphia (1972)
23. Hahn, H.T., Kim, R.Y.: Proof Testing of Composite Materials. *J. Comp. Mater.* 9, 297–311 (1975)
24. Yang, J.N., Jones, D.L.: Load Sequence Effects on Graphite/Epoxy $[\pm 35]_{2S}$ Laminates. In: *Long-Term Behaviour of Composites*, pp. 246–262. ASTM STP 813, Philadelphia (1983)
25. Rotem, A.: Fatigue and Residual Strength of Composite Laminates. *Eng. Fract. Mech.* 25, 819–827 (1986)
26. Adam, T., Dickson, R.F., Jones, C.J., Reiter, H., Harris, B.: A Power Law Fatigue Damage Model for Fibre-Reinforced Plastic Laminates. *Proc. Inst. Mech. Eng., Part C: Mech. Eng. Sci.* 200, 155–166 (1986)
27. Caprino, G., D'Amore, A.: Flexural Fatigue Behaviour of Random Continuous-Fibre-Reinforced Thermoplastic Composites. *Comp. Sci. Tech.* 58, 957–965 (1998)
28. Whitworth, H.A.: Evaluation of the Residual Strength Degradation in Composite Laminates Under Fatigue Loading. *Comp. Struct.* 48, 261–264 (2000)
29. Bergmann, H.W., Prinz, R.: Fatigue Life Estimation of Graphite/Epoxy Laminates Under Consideration of Delamination Growth. *Int. J. Numer. Methods* 27, 323–341 (1989)
30. Feng, X., Gilchrist, M.D., Kinloch, A.J., Matthews, F.L.: Development of a Method for Predicting the Fatigue Life of CFRP Components. In: Degallaix, S., Bathias, C., Fougeres, R. (eds.) *Proceedings of the International Conference on Fatigue of Composites*, pp. 407–414. La Societe Francaise de Metallurgie et de Materiaux, Paris (1997)
31. Hernaff-Gardin, C., Lafarie-Frenot, M.C., Goupillaud, I.: Prediction of Cracking Evolution Under Uniaxial Fatigue Loading in Crossply Composite Laminates. In: Degallaix, S., Bathias, C., Fougeres, R. (eds.) *Proceedings of the International Conference on Fatigue of Composites*, pp. 189–196. La Societe Francaise de Metallurgie et de Materiaux, Paris (1997)
32. Schon, J.: A Model of Fatigue Delamination in Composites. *Comp. Sci. Tech.* 60, 553–558 (2000)

33. Highsmith, A.L., Reifsnider, K.L.: Stiffness-Reduction Mechanisms in Composite Laminates. In: Reifsnider, K.L. (ed.) *Damage in Composite Materials*, pp. 103–117. ASTM STP 775, Philadelphia (1982)
34. Hashin, Z.: Analysis of Cracked Laminates: A Variational Approach. *Mech. Mater.* 4, 121–136 (1985)
35. El-Mahi, A., Berthelot, J.M., Brillaud, J.: Stiffness Reduction and Energy Release Rate of Cross-Ply Laminates During Fatigue Tests. *Comp. Struct.* 30, 123–130 (1995)
36. Smith, P.A., Ogin, S.L.: On Transverse Matrix Cracking in Cross-Ply Laminates Loaded in Simple Bending. *Composites: Part A* 30, 1003–1008 (1999)
37. Katerelos, D.T.G., Kashtalyan, M., Soutis, C., Galiotis, C.: Matrix Cracking in Polymeric Composites Laminates: Modelling and Experiments. *Comp. Sci. Tech.* 68, 2310–2317 (2008)
38. Reifsnider, K.L.: The Critical Element Model: A Modeling Philosophy. *Eng. Fract. Mech.* 25, 739–749 (1986)
39. Sendeckyj, G.P.: Life Prediction for Resin-Matrix Composite Materials. In: Reifsnider, K.L. (ed.) *Fatigue of Composite Materials*. Composite Material Series, vol. 4, pp. 431–483. Elsevier, Amsterdam (1990)
40. Laws, N., Dvorak, G.J., Hejazi, M.: Stiffness Changes in Unidirectional Composites Caused by Crack Systems. *Mech. Mater.* 2, 123–137 (1983)
41. Talreja, R.: Transverse Cracking and Stiffness Reduction in Composite Laminates. *J. Comp. Mater.* 19, 355–375 (1985)
42. Talreja, R.: Stiffness Properties of Composite Laminates with Matrix Cracking and Interior Delamination. *Eng. Fract. Mech.* 25, 751–762 (1986)
43. Allen, D.H., Harris, C.E., Groves, S.E.: A Thermomechanical Constitutive Theory for Elastic Composites with Distributed Damage - I: Theoretical Development. *Int. J. Sol. Struct.* 23, 1301–1318 (1987)
44. Allen, D.H., Harris, C.E., Groves, S.E.: A Thermomechanical Constitutive Theory for Elastic Composites with Distributed Damage - II: Application to Matrix Cracking in Laminated Composites. *Int. J. Sol. Struct.* 23, 1319–1338 (1987)
45. Coats, T.W., Harris, C.E.: A Progressive Damage Methodology for Residual Strength Predictions of Notched Composite Panels. *J. Comp. Mater.* 33, 2193–2224 (1999)
46. Spearing, S.M., Beaumont, P.W.R., Smith, P.A.: Fatigue Damage Mechanics of Composite Materials - Part IV: Prediction of Post-fatigue Stiffness. *Comp. Sci. Tech.* 44, 309–317 (1992)
47. Ladeveze, P., LeDantec, E.: Damage Modeling of the Elementary Ply for Laminated Composites. *Comp. Sci. Tech.* 43, 257–267 (1992)
48. Shokrieh, M.M., Lessard, L.B.: Progressive Fatigue Damage Modeling of Composite Materials, Part I: Modeling. *J. Comp. Mater.* 34, 1056–1080 (2000)
49. Shokrieh, M.M., Lessard, L.B.: Progressive Fatigue Damage Modeling of Composite Materials, Part II: Material Characterization and Model Verification. *J. Comp. Mater.* 34, 1081–1116 (2000)
50. Camanho, P.P., Matthews, F.L.: A Progressive Damage Model for Mechanically Fastened Joints in Composite Laminates. *J. Comp. Mater.* 33, 2248–2280 (1999)
51. Sun, C.T., Vaidya, R.S.: Prediction of Composite Properties from a Representative Volume Element. *Comp. Sci. Tech.* 56, 171–179 (1996)
52. Xia, Z., Chen, Y., Ellyin, F.: A Meso/Micro-mechanical Model for Damage Progression in Glass-Fiber/Epoxy Cross-Ply Laminates by Finite-Element Analysis. *Comp. Sci. Tech.* 60, 1171–1179 (2000)

53. Song, S., Waas, A.M., Shahwan, K.W., Xiao, X., Faruque, O.: Braided Textile Composites Under Compressive Loads: Modeling the Response, Strength and Degradation. *Comp. Sci. Tech.* 67, 3059–3070 (2007)
54. Guo-Dong, F., Jun, L., Bao-Lai, W.: Progressive Damage and Nonlinear Analysis of 3D Four Directional Braided Composites Under Uniaxial Tension. *Comp. Struct.* 89, 126–133 (2009)
55. Mishnaevsky Jr., L., Brondsted, P.: Micromechanics of Damage in Unidirectional Fiber Reinforced Composites: 3D Computational Analysis. *Comp. Sci. Tech.* 69, 1036–1044 (2009)
56. Lubineau, G., Ladeveze, P., Violeau, D.: Durability of CFRP Laminates Under Thermomechanical Loading: A Micro-Meso Damage Model. *Comp. Sci. Tech.* 66, 983–992 (2006)
57. Lubineau, G., Violeau, D., Ladeveze, P.: Illustrations of a Microdamage Model for Laminates Under Oxidizing Thermal Cycling. *Comp. Sci. Tech.* 69, 3–9 (2009)
58. Shi, Z., Zhang, R.: Fatigue Damage of Fiber Reinforced Composites: Simultaneous Growth of Interfacial Debonding and Matrix Annular Crack Surrounding Fiber. *J. Comp. Mater.* 42, 2247–2258 (2008)
59. Material Property Data Sheets, Matweb, <http://www.matweb.com>
60. Blassiau, S., Thionnet, A., Bunsell, A.R.: Micromechanics of Load Transfer in a Unidirectional Carbon Fiber-Reinforced Epoxy Composite Due to Fiber Failures - Part I: Micromechanisms and 3D Analysis of Load Transfer: The Elastic Case. *Comp. Struct.* 74, 303–318 (2006)
61. Ha, S.K., Jin, K.K., Huang, Y.: Micro-mechanics of Failure (MMF) for Continuous Fiber Reinforced Composites. *J. Comp. Mater.* 42, 1873–1895 (2008)

Parallelizing Power Systems Simulation for Multi-core Clusters: Design for an SME

Hossein Pourreza¹, Ani Gole², Shaahin Filizadeh², and Peter Graham¹

¹ Department of Computer Science, University of Manitoba
Winnipeg, MB, Canada R3T 2N2

² Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, MB, Canada R3T 2N2

Abstract. In this paper, we discuss the parallelization of power systems simulation using modern clusters consisting of multi-core compute nodes interconnected by a low latency interconnect. We describe an implementation strategy that exploits three types of parallelization that are amenable to the various types of inter-core “connectivity” commonly seen in such clusters. We also consider a set of design criteria that are important to Small-Medium Enterprises (SMEs) and how they affect implementation strategy. We report the results of some initial experiments done using components that will form a part of the prototype system under construction and then discuss the expected performance gains for the entire system based on manpower invested. We also identify several key factors in design for SMEs that differentiate their parallel systems needs from those of large organizations and research scientists.

1 Introduction

The design parameters of a power system are usually optimized (minimized or maximized) through simulating the system and examining its behavior. Electromagnetic Transient (EMT) simulation [1] is one widely-used type of power systems simulation. Traditionally, a design engineer manually examines the results of simulation and, if required, changes a set of parameters and repeats the simulation. This process continues until a desired goal is achieved. Recent developments in simulation systems try to automate this time-consuming process of manually inspecting the results and repeating the simulation by incorporating an optimization component in the simulation system [2]. After running the simulation using a given set of parameters, an Objective Function (OF) is calculated. The OF penalizes poor system parameters and new parameters are selected to improve the OF. This process continues until an “optimal” OF value is obtained. The need for very short time steps (on the order of a micro second) and to repeat the simulation many times to achieve an optimal OF, increases the time required for even a simple simulation with only a few parameters. To speedup the simulation process and to enable the simulation of large power systems, the use of parallel computing is attractive. Parallel processing techniques can be generally applied to simulate power systems at, at least, three levels:

1. Exploring the problem space in parallel to find an optimal solution
2. Incorporating parallel algorithms in the simulation engine to speedup key sequential operations (e.g. matrix-vector multiplication, etc.)
3. Decomposing large power systems into smaller ones and simulating the smaller parts in parallel

In the first approach, multiple simulations are performed concurrently with different parameters. Once a set of simulations is finished, the OFs are collected and the “best” one is selected. If the calculated OF is judged to be sub-optimal, another set of simulations are performed, and the process continues. This first approach is straightforward, easily parallelizable, and requires minimal change to the underlying simulation engine (in our case, the EMTDC [3] software). It is also well suited to methods such as Genetic Algorithms (GA) [4], multi-resolution mesh [5], and gradient-based algorithms [6], etc. These methods are largely compute bound and require only minimal communication to distribute simulation parameters and to gather OF results for comparison.

In the second approach, the key sequential algorithms of the simulation software (e.g. matrix-vector multiplication, matrix inversion, etc.) must be replaced with their parallel counterparts. This approach requires fundamental modifications to the existing simulation software and, in naive parallel implementations, may suffer significant communications/synchronization overhead.

Finally, in the third approach, a large power system can be decomposed into smaller ones. Using the inherent propagation delays between decomposed components in a geographically distributed system, different parts can be simulated in parallel, and the result of one can be fed into the computation of another at a later time step.

With the increasing prevalence and scale of multi-core computers and with the rapidly decreasing cost of low latency interconnects such as infiniband, the attractiveness of small-scale clusters for power systems simulation is growing. The parallelization approaches, discussed earlier, are compatible with a well-connected cluster of multi-core computers in which different simulations can be assigned to different nodes and the simulation software can make use of the multiple-cores within a node to perform its task faster.

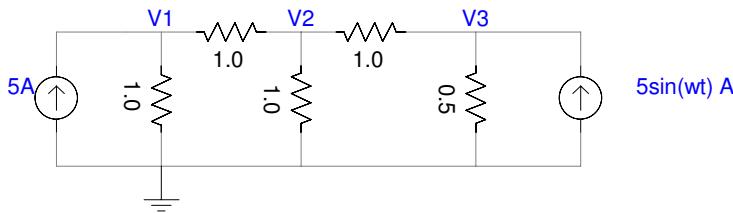
Exploiting the first approach, we have parallelized a number of existing optimization-enabled simulations, including some using techniques based on multi-resolution mesh, heuristic search, and gradient descent. We have also parallelized a small number of key algorithms used by EMTDC for multi-core execution. Our initial results are promising, and we are in the process of integrating all three approaches in a simulation framework targeted at supporting power systems design for a range of large and potentially complex circuits. A complicating factor in this though is the need to produce a parallel solution that meets the needs of a small/medium enterprise (SME) that would be likely to market such software commercially. In traditional parallelization done by academic researchers, “heroic” programming efforts are often involved with large investments in time. Further, such implementations are often highly tuned to specific target platforms. Neither of these are acceptable to SMEs who typically have limited

resources, tight timelines, and whose code must a) be inexpensive to maintain and b) have longevity across a continuum of platforms. Such challenging business constraints faced by SMEs have been documented (e.g. [7]). In doing this work, we have focused on developing code for a relatively generic platform expected to be ubiquitous and to persist for some time, and we have paid attention to the effort involved in developing code. As a result, we expect to see practically useful speedups using the framework we have developed on small, highly cost-effective clusters that can be easily deployed and used by power systems design engineers.

In the rest of this paper we first briefly describe the power systems simulation problem in Section 2. In Section 3 we describe our proposed parallelization framework and motivate its appropriateness both for the power systems simulation problem and current cluster technology. We present our experiences to date with a number of framework components we have built and present some early speedup results in Section 4. We then discuss the suitability of our work for an SME environment in Section 5. Finally, in Section 6, we provide our conclusions and discuss directions for future work.

2 Power Systems Simulation

The simplest form of power systems have only passive elements (i.e., resistors, capacitors and inductors) and power sources (i.e., voltage sources and/or current sources). In such a circuit, currents are assumed to be known and voltages are



(a) Simple Circuit

$$\underbrace{\begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 3 \end{bmatrix}}_Y \underbrace{\begin{bmatrix} V1 \\ V2 \\ V3 \end{bmatrix}}_V = \underbrace{\begin{bmatrix} 5 \\ 0 \\ 5\sin(wt) \end{bmatrix}}_J$$

(b) Admittance Matrix Equation

Fig. 1. Simple Circuit and Admittance Matrix Equation

unknowns. One well known and widely-used method to solve this type of circuit is the “admittance matrix” method [8].

In this method, all components are converted to resistors and current sources. Some of the converted components are time varying so the value of a component at time t is dependent on its value at time $t - 1$. To solve a circuit with time varying components, we must choose a small time step and solve the circuit at each time step until the system stabilizes. To solve a circuit, a matrix equation (the admittance matrix equation) is formed. This equation can easily be automatically generated from a circuit netlist which makes this method suitable for use in a power systems simulation program.

Figure 1 shows a simple circuit and its corresponding matrix equation (construction of the equation is outside the scope of this paper). To solve the matrix equation, we generally need to invert the Y matrix and do a matrix-vector multiplication. The dimension of the Y matrix is determined by the number of unknown voltages. Thus, bigger circuits (with more unknown voltages) will have bigger Y matrices and the memory requirements and computational cost of solving the matrix equation will increase. For large systems, it is thus attractive to decompose the Y matrix and use parallel matrix algorithms to solve the admittance matrix equation.

3 Simulation Framework

Most practitioners currently use single workstations to design and simulate power systems. Commercial versions of simulation programs like PSCAD/EMTDC [9] provide engineers with convenient graphical interfaces to describe the circuits being analyzed and to interpret the results of simulations. Unfortunately though, execution of the underlying simulation engine is normally restricted to a single workstation processor. Even with the advent of multi-core processors, existing simulation engines have seen little benefit since they are not inherently designed to exploit multiple cores. Given that the prevalence and scale of multi-core processors are growing and seem to be likely to do so for a number of years to come, it is no longer practical to ignore the potential use of parallelism.

The traditional serial workstation approach has two clear drawbacks. First, it makes the process of doing “design space exploration” tedious. Running many simulations to optimize design parameters is slow and requires direct engineer involvement. It would be beneficial to automate and parallelize this process as much as possible. Second, for large circuits, the computational overhead of solving the admittance matrix equations makes simulation very slow. Even coarse-grained parallelization of large simulations would result in small-N speedups¹ which would be a very practical benefit to practicing engineers.

The parallelization of power systems simulation has historically been limited by the cost and complexity of building and using parallel computers. Further, aggressive parallelization attempts have met with little success due to tight coupling between parallel processes requiring frequent communication over slow

¹ Speedups, typically, of at most tens of times.

links. A number of recent developments/trends, however, now make a practical parallel implementation more feasible.

Early shared-memory parallel computers were simply cost-prohibitive for doing power systems simulations. Many practitioners' problems were relatively small, and thus, the high cost of a shared memory machine was hard to justify except for a small user subset. This meant that a serial version of the simulation software would have to be maintained as well as a parallel version. Given the small number of users doing power systems simulation, the cost of parallelizing the software and of maintaining two code bases was highly undesirable, particularly for an SME.

Early cluster systems solved the cost issue with using shared memory machines but failed to offer attractive performance. Using standard interconnects, the speedups that could be obtained were disappointing. The development of low latency interconnects (e.g. Myrinet [10]) offered better potential performance but such networks remained special-purpose and thus expensive. Again, cost and complexity issues and the need to maintain separate code bases precluded parallelization efforts.

The prevalence of inexpensive, commodity multi-core processors, the availability of large, inexpensive memories and the rapid decrease in cost of near-commodity Infiniband [11] networks has now resulted in a parallel computing environment that is much better suited to power systems design and analysis. We now see systems available with quad processors, each of which contain four or six cores and we expect such systems with eight cores shortly. Further, these machines support large memory capacities and fast memory sharing with minimal NUMA penalties. Additionally, quad data rate (QDR) infiniband will provide low latency and higher bandwidth interconnections between nodes in clusters built from such machines. Such clusters will provide a relatively generic (with respect to processor type and network technology), cost-effective, and likely long-lived platform on which to intelligently parallelize power systems design and analysis.

Our goal is to create an effective parallel power systems design and analysis framework that can provide useful speedups on practical problems without huge development efforts using relatively inexpensive multi-core/infiniband clusters while maintaining a single code base. Our framework will include a feature to help automate design space exploration as well as to provide parallelism-based speedup of individual simulations for both small and large simulation tasks. Our approach is to exploit the three forms of parallelism described earlier in this paper. By doing so, we believe we can produce a single parallel code base that will support a range of application uses and run on machines ranging from a single, multi-core machine to various sizes of infiniband (or similar high bandwidth, low-latency network) connected clusters of such machines.

Our initial prototype uses a fixed decomposition strategy for parallel execution. We restrict the use of multiple cluster nodes to design-space exploration tasks. Such tasks are driven by heuristic mechanisms such as genetic algorithms and simulated annealing which are embarrassingly parallel and, therefore, well suited to distribution across cluster nodes. We have implemented a number of

such heuristics running over MPI providing input to, and using results from, EMTDC to calculate the needed objective function. In systems with only a single node, an MPI hosts file with a single entry (or perhaps a few duplicate entries) for the single node can be used so the MPI part of the code base need not be changed for this special case. This will allow continued use of desktop platforms for small-scale design problems.

The multiple cores on each node are used to speed up the necessary repeated solutions of the admittance matrix equation. This involves using small-scale shared-memory code that exploits the multi-core nature of each cluster node to do matrix-vector multiplication and matrix inversion in parallel. We have implemented and tested this code using OpenMP and with very little effort have achieved reasonably good scalability up to six or eight cores on test hardware. (Such low-effort, small-scale speedups are attractive and of immediate use to SMEs.) No support has yet been provided in the prototype to support the decomposition of large circuits though we do have an implementation of simple transmission lines which would be logical points of decomposition for large circuits. Our approach obviously results in a hybrid (MPI-OpenMP) solution when doing design space exploration.

Shortly, we will add support for decomposing large circuits to our prototype to meet the growing demand to simulate larger power systems. Our prototype will then need to incorporate dynamic adaptation to be able to choose to use available cluster nodes for a combination of design space exploration and simulation of circuit “partitions”. Adaptation will be based on circuit size and number of available compute nodes. Being able to do this will also help to ensure that the memory requirements for simulating large circuits can be more effectively met, collectively, by multiple cluster nodes. We will implement this adaptivity at the MPI level to, again, ensure that it is possible to use a single code base. With the recent emphasis placed on efficient memory-based MPI communications, we expect this to be a practical approach, even if used on single processors with larger core counts as are predicted in the near future.

4 Initial Experiences

Building the entire framework for power systems design and analysis is a major undertaking. We are currently creating key components and evaluating the success of different parallelization strategies. To date, we have parallelized a number of design space exploration techniques using MPI and some key algorithms required to parallelize the simulation process itself using OpenMP. We are assembling these components to produce the prototype framework as the components become available and stable. In this section, we report on the status of our work to date and provide some basic speedup results for the implemented components. Note that all components have been designed given constraints that are typical for SMEs. In particular, our design motto has been “not fastest, but faster, quickly”. This reflects the fact that SMEs forced to adopt parallelism due to the spread of multi-core machines don’t care about the fastest solution (via

heroic programming efforts) but do want some useful speedup, and they need to achieve such speedups quickly and without significant human costs.

In all results presented, speedup is calculated in the normal way as $S_p = \frac{T_1}{T_p}$ where p is the number of processors/cores used, T_1 is the serial execution time (not the time for a parallel run on a single processor/core) and T_p is the parallel execution time on p processors/cores.

4.1 Design Space Exploration

In this subsection, we present our experiences with our parallel MPI-based implementation of some key power system simulation tools for design space exploration. These tools include optimization techniques for single and multi objective functions and techniques for sensitivity analysis and are assessed using real-world simulation problems.

Test Environment. We tested our code on a cluster of 10 SunFire 4200s, each having two dual-core AMD Opteron processors running at 2.6 GHz and 8 GB of RAM giving a total of 40 cores. The interconnection was standard data rate Infiniband. We used OpenMPI over Infiniband. Time measurements were taken using functions embedded in our code. Each program was run multiple times and the average of running times are reported. The variances are insignificant.

Genetic Algorithm Optimization. Genetic algorithms can be used to search a problem space to find an optimal solution. The method starts with a set of random possible solutions (the initial population) and the *fitness* of each solution is evaluated using an Objective Function (OF) called the *fitness* function. The OF penalizes *unfit* solutions and fit ones are collected in a new set (the next generation). Certain genetic operators (mutation, crossover) are used to introduce “population” variability.

We adapted the MPIKAIA [12] code to find an optimal switch configuration for a Space-Vector Modulated (SVM) converter. In these converters there are different states based on the on/off status of their controlled switches. The goal is to find a combination of states to get a desired output voltage waveform. The OF therefore assesses waveforms and a circuit simulation is required to evaluate the OF for a given switch configuration. To find an optimal configuration, a genetic algorithm is used to “automatically” evaluate many possible configurations. By running the required multiple simulations in parallel, we were able to achieve good speed up in addition to relieving the design engineer of the tedium of overseeing each simulation and selecting new simulation parameters. We created an initial population of size 40 (one per available core) and assigned one fitness evaluation (i.e. simulation) to each processor. Figure 2 shows the running times of the sequential and parallel versions of the SVM controller optimization. The achieved speedup was 13.34 despite the independence of the simulations. The primary reason for the lower speedup was the existence of some sequential code (e.g. comparison of results and selection of the new population) which could not be done in parallel. Some overhead is also attributable to distribution and collection of data over the Infiniband interconnect.

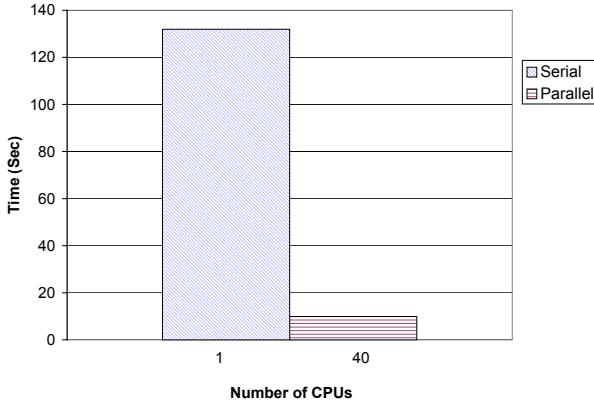


Fig. 2. Serial and Parallel Times for the SVM Converter

Gradient-based Optimization. In a typical gradient-based optimization technique [6], first order derivatives of the OF are numerically calculated to determine the gradient of the OF. For an OF with n parameters, $2n$ derivatives must be calculated. The calculated gradient is used to determine in which direction the new set of parameters must be generated. Once the gradient is calculated, a step length must also be calculated which requires another S evaluations of the OF. Selecting a good step length can result in reaching the optimal point faster. If the parameters of the OF have boundary constraints, those must be considered in determining the step length. The process of moving toward the optimal solution using the calculated step length continues (for I iterations) until an optimal solution is found. In total, $(2n + S) * I$ evaluations of the OF (i.e. simulation runs) are required to find an optimal solution. To make this process faster, parallel computing can be used where evaluations of the OF are assigned to each of available cores. Assuming that we have $2n$ cores and also $S < 2n$, the total calculations are reduced to $(1+1)*I$. In this ideal case (ignoring overhead), there will be a speedup of $n + S/2$.

We selected a control system for a Static Synchronous Compensator (StatCom) [1] as an example problem for parallel gradient-based optimization. The StatCom problem has a relatively simple OF with five input parameters. To calculate the first order derivatives of the OF, 10 calculations were therefore required. Based on experiments with the serial version of the program, we found that, on average, 4 calls to the objective function were required to compute a step length. We therefore expected an ideal speedup of $n + s/2 = 5 + 4/2 = 7$. Our experiments, using an adequate number of CPUs (14 in this case) show a speedup of 6.81 in practice which is very close to the ideal. The running times of both the serial and parallel versions for the StatCom example problem are shown in Figure 3.

Sensitivity Analysis. Sensitivity analysis examines the sensitivity of an optimal solution to variation of parameters. Depending on the shape of an OF,

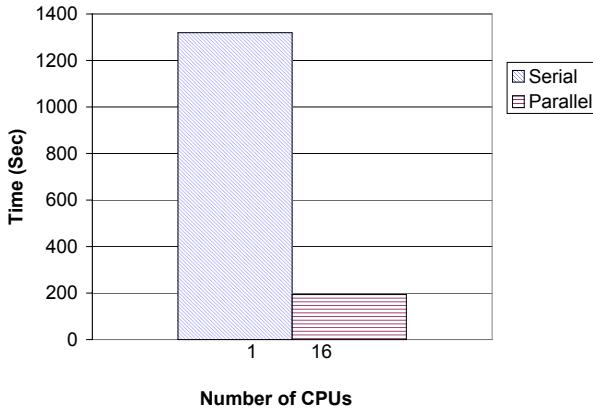


Fig. 3. Serial and Parallel Times for StatCom Example

very small variations in the parameters from an optimal point can cause a big difference in the value of the OF. First - order derivatives do not accurately measure the sensitivity of an OF [1] so second order derivatives of a function are used to correctly measure the sensitivity. It can be shown that the total required computation for calculating the second order derivatives of a function with n parameters is $n(n + 1)$ [1]. Thus, if M CPUs are available, in the ideal case $\min(M, n(n + 1))$ speedup is expected.

We measured the sensitivity of our StatCom problem to variation in its parameters using the second-order derivatives of the objective function. With 5 parameters, we need 30 evaluations of the OF. Given the required number of CPUs (30 in this example), we expected a speedup of 30. However, we have a call to the OF before starting the parallel computation, and this increases the total computation time considerably. Thus, in practice we achieved a speedup of only 14.35. Figure 4 shows the running times of both the serial and parallel versions of the program for calculating the sensitivity of the StatCom example.

Pareto Frontier. Some problems in power systems have more than one parameter that must be simultaneously optimized. These multi-objective problems can have several *competing* objectives where optimizing one parameter can make the others worse. A Pareto optimum is a set of parameters where none of the corresponding objectives can be optimized without making at least one of them worse.

An interesting problem is finding the set of all Pareto optima. This is known as the *Pareto frontier*. One method to find the Pareto frontier is by combining different objectives into one using different weights [1]. By trying different weights, all objectives will have a chance to take part in the combined objective. This method, however, requires multiple evaluations of the function and trying different weights.

Two levels of parallelism are possible for this problem. The first (lower) level is finding the optimum value of the OF in parallel (e.g. using the parallel

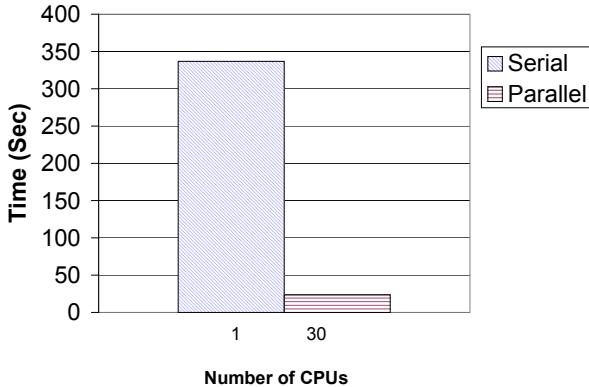


Fig. 4. Serial and Parallel Run Times for Calculating the Sensitivity of the Objective Function for the StatCom Example for its Parameters

gradient-based optimization). The second (higher) level is computing the combined function with different weights in parallel. These two levels are depicted in Figure 5 as *slave_i* and *master_j* nodes respectively. The *root* node initializes and later terminates the whole process. Having these two levels requires more CPUs if we are to *fully* parallelize the problem. If the number of CPUs is less than the number of tasks, a scheduling technique (e.g. round robin) must be used to distribute work to the cores. Recall that we need $2 * n * S * I$ calculations to find an optimal value of a function using gradient-based optimization. Thus, if we have W different weights to calculate (these will be Pareto points), the total computational cost of finding the Pareto frontier will be $W * 2 * n * S * I$.

As an example, we chose to calculate the Pareto frontier of a function with 2 objectives and we selected 9 different weights to obtain 9 Pareto points. The simplicity of this objective function resulted in low volume of computation that did not justify using parallel computing. Furthermore, the serial version calculates each Pareto point using the previous Pareto point as its starting point which decreases the total running time. In the parallel version, on the other hand, each Pareto point was calculated using the initial point. For these two reasons we

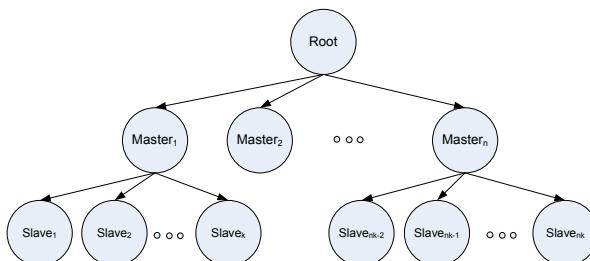


Fig. 5. Organization of Cores to fully Parallelize the Finding of Pareto Points

noticed a sharp increase in the computation time when we attempted to parallelize computation of the Pareto frontier for this simple function. Figure 6 shows the running times of the serial and parallel versions of the program finding Pareto points. Clearly, caution is required in using this technique in parallel. This illustrates how parallelizations that appear attractive to naive parallel programmers may not be useful. In this case, another technique such as vector evaluated particle swarm optimization [13] might have proved to be more effective. Providing a set of available tools that can be experimented with by SME engineers would offer a simple and useful solution to the problem. Also, developing parallel code that is scale-aware so it can run serially when the size of the problem is too small will be very useful.

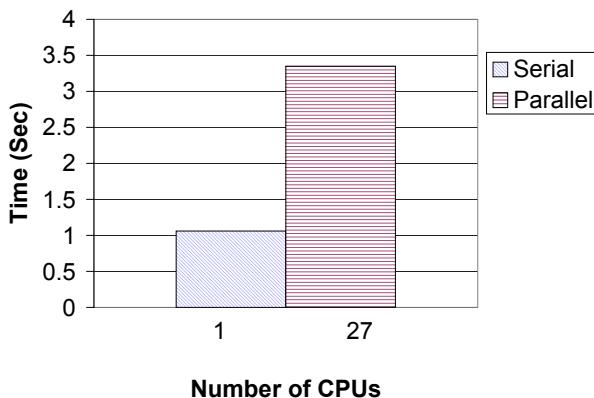


Fig. 6. Running times of parallel and serial approaches to finding Pareto points

4.2 Simulation Parallelization

The two fundamental operations involved in solving admittance matrix equations are matrix vector multiply and matrix inversion. We implemented and assessed simple shared-memory parallel versions of both of these operations. The matrix vector multiply was implemented by partitioning the Y matrix by rows and distributing the work across the available cores. The matrix inversion was implemented using a naive algorithm based on Gaussian elimination with pivoting where parallelism was introduced when “knocking out” matrix elements in rows beneath and above the unitized pivot.

Test Environment. We tested our OpenMP code on a SunFire 4600 with eight dual-core AMD Opteron processors running at 2.6 GHz and 32 GB of RAM. Time measurements were taken using functions embedded in our code. Each program was run multiple times and the average of running times are reported.

Speedup Results. Figure 7 shows the execution times when doing a matrix vector multiply on an 800x800 matrix. Good speedup results are obtained using up to six cores. Some small additional speedup is obtained with up to eight cores.

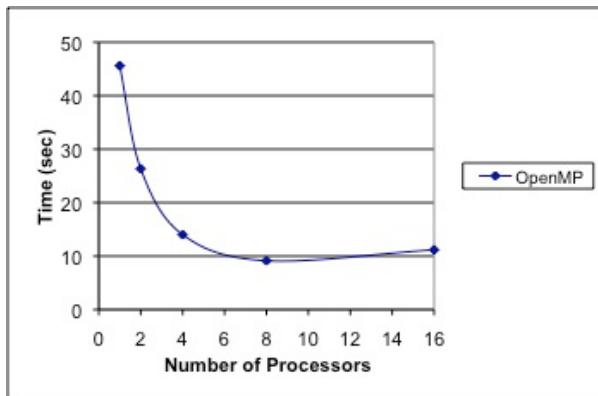


Fig. 7. Running time of OpenMP Matrix Vector Multiply

Figure 8 shows the execution times for matrix inversion (via Gaussian elimination) on a 500x500 matrix. Even using this naive algorithm, reasonable speedups are obtained using up to four cores with minor improvements up to eight.

Note that the SunFire 4600 uses a motherboard-daughterboard configuration, and in other work, we have seen overheads when accessing memory across the boundary between the boards. As each board houses four, dual core processors, this may explain why noticeable slowdowns are seen after about eight cores.

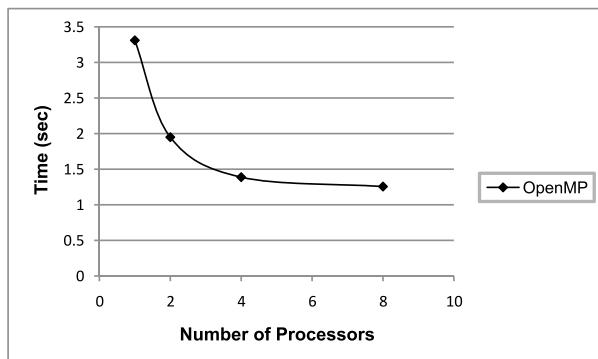


Fig. 8. Running time of OpenMP Matrix Invert

5 Parallel Design for an SME Environment

With the advent of multi-core systems, it is becoming impractical for SMEs to continue to sustain a purely serial application. Adding cores to a chip requires additional “silicon real-estate” which decreases the space available to optimize

performance in a single core (e.g. via instruction level parallelism). This means that in the relatively short term, the performance of purely serial applications will begin to decrease on new hardware. As such, it is now (or will very soon be) necessary for software vendors to re-engineer their code to exploit the small-scale parallelism opportunities afforded by multi-core processors just to maintain the performance they have been able to offer in the past. For SMEs, in particular, this is a challenging task due to limited personnel resources, potential lack of expertise and tighter constraints on product development and enhancement cycles.

A large part of the reluctance of SMEs to consider parallel systems as platforms for their software products has been the need to maintain two code bases: one for the serial version of their code and another for the corresponding parallel version. The historical need to have two code bases is clearly illustrated in power systems simulation where the distribution of end users of such software has tended to be heavily tailed with many users working on relatively small circuits that could be adequately simulated using desktop environments and considerably fewer requiring the use of special purpose parallel machines. Since the bulk of the market for power systems simulation code did not need parallelization, it made no sense to provide a single parallel code base.

The pending change to multi-core processors offers an opportunity to SMEs, including those producing power systems simulation software. Since a change to some sort of parallel code base is now necessitated, some minor investment in intelligent design (as suggested in this paper) could lead not only to sustained but also to enhance performance as well as to a long-term “parallel framework” that will allow a single code base to exploit a range of scales of parallel machines to meet the needs of both low-end and high-end customers. This must be done, however, subject to the constraints imposed by SME development environments.

In general, commercial SME software vendors are less interested in achieving the best possible parallel speedup than they are in obtaining some speedup, cost-effectively. This reflects a broader perspective where any investment in effort must yield overall return for the SME, not just their end customers. Employee costs dominate both development and maintenance of code so parallelization must be relatively straightforward. Expensive, “heroic” programming efforts to squeeze every bit of performance out of a parallel algorithm are not cost-effective. Instead, applying simple parallelization strategies that are easy to implement can yield return on investment in terms of enhanced sales and retained customers.

SME software vendors must also target ubiquitous, long-lived execution platforms. Specializing their code to a little used or short-lived parallel machine is unattractive as SMEs cannot incur the cost of frequent code modification/porting. Instead, a platform is needed that will appeal to as broad a subset of an SME’s user base as possible and which also is likely to be used for many years.

The parallel design described in this paper meets both these key constraints. Our code is designed to run on readily available, flexible and widely accepted “commodity” platforms. Further, given trends in the industry and the factors driving those trends, it is safe to assume that clusters of well-connected multi-core processors are a platform that will exist for some time. Also, our choice to

use straightforward, easily parallelizable, techniques reflects the expected limitations of parallel code design and implementation in an SME environment. Our implementation times were measured in person hours or days rather than person months or years. Further, the use of OpenMP for shared memory parallelization meant that the changes required to *existing* serial code were minimal (only a handful of OpenMP pragmas) and this approach is highly desirable to support fast conversion from an existing serial code base to a simple parallel one.

Additionally, using well understood parallel coding techniques and widely used parallel standards (MPI and OpenMP), our code can be successfully and productively run, from a single code base, on a range of multi-core cluster machines offering small scale speedups on single (multi-core) processors and simple problems and larger (tens of times) speedups on clusters for design space search problems. These sorts of speedups are useful for practicing power design engineers specifically, and for SME software vendors and users generally.

6 Conclusions and Future Work

The simulation of power systems is an area where speedup through parallelization could provide significant benefits. Recent trends enabling cost-effective multiple processor, multiple core machines with large memories combined with significant decreases in the price of Infiniband low-latency, high bandwidth interconnects provide the basis for an attractive cluster-based solution that suits the needs of power systems simulation. In this paper, we have described a framework for the design and evaluation of power systems that exploits simple types of parallelization for a hybrid MPI-OpenMP implementation while supporting a single, parallel, code base. The possibility of maintaining a single code base significantly increases the attractiveness of parallelizing such simulations in commercial settings. We have also described the results of some experiments we have done with key components that will form parts of the final framework. Through the careful application of limited parallelism at three distinct levels, we believe we can achieve useful speedups on a range of related platforms (from single multi-core machine to Infiniband clusters) that will provide benefits to practicing engineers.

In the short term, we will continue to integrate the components we have built to create a complete operational prototype. This will include code to assist users in easily creating objective functions for design space applications. Our next step will be to add the ability to decompose large circuits for parallel simulation in parts. This will be followed by the creation of algorithms to enable dynamic adaptation so the framework can automatically use available parallel resources to most effectively meet the needs of the work being done.

References

1. Heidari, M., Filizadeh, S., Gole, A.M.: Support tools for simulation-based optimal design of power networks with embedded power electronics. *IEEE Transactions on Power Delivery* 23(3), 1561–1570 (2008)

2. Gole, A.M., Filizadeh, S., Menzies, R.W., Wilson, P.L.: Optimization-enabled electromagnetic transient simulation. *IEEE Transactions on Power Delivery* 20(1), 512–518 (2005)
3. EMTDC: User's Guide. Manitoba HVDC Research Center (2003)
4. Haupt, R.L., Haupt, S.E.: Practical Genetic Algorithms. John Wiley and Sons, New York (1998)
5. Kobravi, K.: Optimization-enabled transient simulation for design of power circuits with multi-modal objective functions. Master's thesis, Electrical and Computer Engineering, University of Manitoba (2007)
6. Chong, E., Kah, P.: An introduction to optimization. John Wiley and Sons, New York (2001)
7. Burgess, S.: Managing Information Technology in Small Business: Issues and Challenges. Idea Group, Hershey (2002)
8. Dommel, H.W.: Digital computer solution of electromagnetic transients in single- and multiphase networks. *IEEE Transactions on Power Apparatus and Systems* PAS-88(4) (1969)
9. PSCAD, <https://pscad.com/products/pscad/>
10. Boden, N., Cohen, D., Felderman, R., Kulawik, A., Sietz, C., Seizovic, J., Su, W.: Myrinet: A Gigabit per second Local Area Network. *IEEE Micro* 15(1), 29–36 (1995)
11. Infiniband Trade Association, <http://www.infinibandta.org/specs/>
12. Metcalfe, T., Charbonneau, P.: Stellar structure modeling using a parallel genetic algorithm for objective global optimization. *Journal of Computational Physics* 185, 176–193 (2003)
13. Parsopoulos, K.E., Tasoulis, D.K., Vrahatis, M.N.: Multiobjective optimization using parallel vector evaluated particle swarm optimization. In: Artificial Intelligence and Applications (2004)

OpenMP Parallelization of a Mickens Time-Integration Scheme for a Mixed-Culture Biofilm Model and Its Performance on Multi-core and Multi-processor Computers

Nasim Muhammad and Hermann J. Eberl

Dept. of Mathematics and Statistics, University of Guelph
Guelph, ON, Canada, N1G 2W1
{nmuhamma,heberl}@uoguelph.ca

Abstract. We document and compare the performance of an OpenMP parallelized simulation code for a mixed-culture biofilm model on a desktop workstation with two quad core Xeon processors, and on SGI Altix Systems with single core and dual core Itanium processors. The underlying model is a parabolic system of highly non-linear partial differential equations, which is discretized in time using a non-local Mickens scheme, and in space using a standard finite difference method.

1 Introduction

Bacterial biofilms are microbial depositions on surfaces and interfaces in aqueous environments. Bacteria attach to the surface (called substratum in the biofilm literature) and produce a matrix of extracellular polymeric substances (a.k.a. EPS). The microorganisms themselves are embedded in this gel-like layer, which protects them against mechanical washout and antimicrobials. Despite their name, biofilm communities do not always form as homogeneous layers but can grow in spatially highly irregular morphological structures. Biofilms are omnipresent in nature; in environmental engineering, biofilm based technologies are developed for wastewater treatment, groundwater protection, soil remediation, etc. In a medical context, biofilms are dangerous, causing bacterial infections. In industrial systems they can accelerate corrosion or system clogging.

Modern mathematical models of biofilms account for the spatial structure of the colony. These are complex two- or three-dimensional models that combine population and resource dynamics, mass transfer and fluid dynamics. In [7] it was argued that many aspects of biofilm modeling are tasks for High Performance Computing. In some cases, this is due to the shear complexity of the aspects studied, e.g., in the case of three-dimensional biofilm-fluid interaction problems that describe mechanical deformation of biofilms which eventually can lead to sloughing of biomass. In other applications, a single simulation can be run on a desktop workstation, but due to the uncertainties that are inherent in most biofilm modeling studies, many such simulations are required in a numerical

experiment. In [9], for example, a simulation experiment of biofilm disinfection in a slow flow field was carried out, that consisted of more than 320 single two-dimensional simulations. While each individual simulation can be conducted on a regular desktop PC in less than one hour computing time, the sheer number of simulations becomes overwhelming. In principle, there are two approaches to deal with this type of problem complexity on parallel computers: (i) run several independent serial simulations concurrently on individual processors [i.e. a serial farm approach], or (ii) run the simulation in parallel and distributed over several processors and cores, or combinations of (i) and (ii). In (i) no communication overhead is involved but the memory requirements scale with the number of simulations/cores used, which also and most critically affects cache utilization and can slow down computation or even make it rather impractical. On the other hand, in (ii) if only a single compute job is running, the memory requirements remain low but computing efficiency is lost due to parallel overhead. Which of these two strategies is best, or how to combine them most efficiently is a priori not known. It depends on the specific problem at hand (type and size) and on the hardware that one wishes to use.

With the current development of multicore processors and the increasing availability of multi-processor desktops computers, parallel computing is entering mainstream [13][16][19][23]. While in the past parallel computers were tools to be used by relatively few researchers, now every computational scientist has to face the challenges that come with it in order to take advantage of the improvements of technology. Parallelism adds complexity to computer programming, and to exploit it fully and efficiently is difficult. It requires a good understanding of both the algorithm and the hardware to be used. Computational scientists, however, usually are not computer scientists but have their core competence in applied mathematics and/or in the area of application. In order for us to use the parallel abilities of modern computing platforms, fairly high-level approaches to parallel programing are desired, i.e. approaches in which the underlying technical (hardware) issues are hidden away from the user, who then can focus on the algorithmic aspects. The parallelism in many problems in Computational Science and Engineering is relatively simple: Problems are often data parallel and can be parallelized on the loop level. Relatively user-friendly approaches to parallelization of such problems have existed over the years, including (automatic) compiler parallelization, High Performance Fortran (for distributed memory machines), or OpenMP (for shared memory architectures). However, on traditional mainframe-like parallel computers, these concepts never gained a dominant role in Computational Science, but remained in the shadow of first PVM and then MPI, which are much more complicated to program, but add more flexibility and possibilities to hand-tune and optimize simulation codes¹. In fact, that the more difficult to learn programing models like MPI were the standard in parallel computation, may be one reason why parallel computing never entered mainstream in many application areas that could have greatly benefited from it. It

¹ One might argue that OpenMP allows much of the same flexibility, but this is at the expense of loosing the the ease-of-use which makes it so attractive.

appears that the situation is changing now with the advent of cheap multi-core and multi-processor desktop workstations. These are shared memory platforms and, therefore, easy to use shared memory programming approaches are naturally gaining interest in the scientific community. In particular, OpenMP has been widely accepted, because it is very easy to apply in such problem where the parallelism is inherently a loop-level parallelism. The literature contains a number of benchmarking studies of OpenMP performance, cf. [24][15][20][21]. These are usually focused on relatively simple, or primitive aspects. Real-world applications in computational science, on the other hand are much more complex and consist of many millions such primitive tasks. How the results of simple benchmarks carry over to such involved applications is not a priori clear. Since no two applications are alike (or easily comparable), benchmarking is difficult and performance evaluation is essentially a matter of case studies. This is also what we present here. Besides evaluating the parallelization gain on multi-core/multi-processor computers, we will also compare the parallel behavior of the simulation code between commodity built multi-core/multi-processor desktop workstations and more traditional parallel compute servers. While the biofilm problem at hand is mathematically peculiar, due to two non-linear, degenerate diffusion effects, it belongs to a fairly general and traditional category of applications, namely the simulation of nonlinear convection-diffusion-reaction problems. Therefore, our result might be characteristic for other problems as well.

2 The Mathematical Model

2.1 Governing Equations

We consider a biofilm that acts as a biobarrier to prevent the propagation of contaminants and, thus, the pollution of groundwater. The biofilm is formed by two species, one of which (e.g. *Burkholderia cepacia*) degrades a pollutant (e.g. TCE), and one of which (e.g. *Klebsiella oxytoca*) does not. Both species compete for one resource (e.g. organic carbon, oxygen, iron). Reaction-kinetics for such a biobarrier biofilm system have been proposed previously in [3], which we combine with the non-linear diffusion biofilm model that was originally proposed in [8].

The biobarrier biofilm model is described in terms of the independent variables substrate concentration S , pollutant concentration P , volume fraction occupied by the neutral biofilm stabilizer X and by the pollutant degrader Y . The following processes are included in the model: (a) Growth of the pollutant degrader Y , during which both pollutant P and substrate S are consumed. (b) Growth of the biofilm stabilizer X during which process substrate S is consumed. (c) Biomass decays due to cell death/lysis. (d) Dissolved substrates S, P are transported by convection in the aqueous region and (e) by diffusion in biofilm and liquid phase. (f) The biomass spreads if the local biomass density reaches the maximal possible cell density. In the two-dimensional computational domain Ω the resulting model reads

$$\begin{aligned}\partial_t S &= \nabla \cdot (d_1(M) \nabla S) - \nabla \cdot (U S_1) - \frac{R_1}{Y_1} - \frac{R_2}{Y_3} \\ \partial_t P &= \nabla \cdot (d_2(M) \nabla P) - \nabla \cdot (U S_2) - \frac{R_2}{Y_2} \\ \partial_t X &= \nabla \cdot (D(M) \nabla X) + R_1 - R_3 \\ \partial_t Y &= \nabla \cdot (D(M) \nabla Y) + R_2 - R_4\end{aligned}\quad (1)$$

where $M(t, x) := X(t, x) + Y(t, x)$ is the total volume fraction occupied by biomass. The biofilm is the region $\Omega_2(t) = \{x \in \Omega : M(t, x) > 0\}$, while $\Omega_1(t) := \{x \in \Omega : M(t, x) = 0\}$ is the aqueous phase without biomass. The reaction terms R_i , $i = 1, \dots, 4$ in (1) are spelled out in Table 1. Constants Y_i , $i = 1, \dots, 3$ indicate how much of the dissolved substrates S and/or P is required to produce biomass. The diffusion coefficients of the dissolved substrates S and P in (1) are reduced inside the biofilm matrix. We make the linearization *ansatz*

$$d_i(M) = d_i(0) + M(d_i(1) - d_i(0)), \quad i = 1, 2, \quad (2)$$

where $d_i(0)$ is the diffusion coefficient in water and $d_i(1)$ in a fully compressed biofilm. Note that thus $d_i(M)$ is bounded from below and above by known constants of the same order of magnitude. Hence, diffusion of S and P behaves essentially Fickian. On the other hand, the density-dependent biomass diffusion coefficient reads [8]

$$D(M) = \delta M^a (1 - M)^{-b}, \quad 0 < \delta \ll 1 < a, b \quad (3)$$

The power law M^a guarantees that biomass does not spread notably if the local biomass fraction is small, i.e. a finite spread of propagation of the biofilm/water interface. The power law $(1 - M)^{-b}$ guarantees that the total volume fraction indeed does not exceed unity [5,10].

Our computational domain is a long skinny channel, mimicking the pore-space between rocks in a soil. For this setup the assumption of creeping flow is in order. Under these restrictions, the flow velocity vector $U = (u, v)^T$ in (1) can be obtained in $\Omega_1(t)$ from the *thin film equations*, cf. [9],

$$p_x = \eta u_{yy}, \quad p_y = 0, \quad u_x + v_y = 0, \quad (4)$$

where η is the dynamic viscosity of the fluid. In $\Omega_2(t)$, $U \equiv 0$. Equations (1) can be solved analytically for our system [9], which allows for a very fast numerical evaluation.

Note that as a consequence of (3) equations (1) are a highly nonlinear, double-degenerate, parabolic system, which poses serious challenges for both analytical and numerical treatment that do not arise in semi-linear diffusion-reaction problems. Model (1) needs to be completed by appropriate initial and boundary conditions. For S, P we specify non-homogeneous Dirichlet conditions on inflow and homogeneous Neumann conditions everywhere else. For the biomass fractions X, Y homogeneous Neumann conditions are prescribed everywhere. Initially S and P are assumed to be constant and the same as the inflow concentration. The biomass fraction X, Y is initially distributed in some randomly chosen pockets along the lower boundary, while $X(0, \cdot) = Y(0, \cdot) = 0$ in the interior of the channel.

Table 1. Reaction kinetics used in (II)

i	process	R_i
1	growth of biofilm stabilizer	$\mu_1 X \frac{S}{\kappa_1 + S}$
2	growth of pollutant degrader	$\mu_2 Y \frac{P}{\kappa_2 + P} \frac{S}{\kappa_3 + P}$
3	decay of biofilm stabilizer	$k_1 X$
4	decay of pollutant degrader	$k_2 Y$

2.2 Numerical Method

The numerical discretization of (II) follows the strategy that was introduced in [6] for a single-species biofilm model. The critical component of every discretization scheme is the treatment of the degenerate biomass equations while standard methods can be used to compute S and P . The key feature in our numerical treatment is a non-local (in time) discretization of the nonlinearities of the model, according to the definition of Nonstandard Finite Difference Schemes [1,17]. In the time-step $t_n \rightarrow t_{n+1}$ we use for the density-dependent diffusive flux

$$D(M) \cdot \nabla X \approx D(M(t_n, \cdot)) \cdot \nabla X(t_{n+1}, \cdot) \quad (5)$$

(and similarly for Y, S, P), for the convective fluxes

$$US \approx U(t_n, \cdot)S(t_{n+1}, \cdot), \quad UP \approx U(t_n, \cdot)P(t_{n+1}, \cdot). \quad (6)$$

The nonlinear reaction terms can be straightforwardly treated in the same manner.

For the numerical realization we introduce a uniform rectangular grid of $n_1 \times n_2$ cells with mesh size Δx . The dependend variables S, P, X, Y are approximated in the cell centers. We denote by $S_{i,j}^k$ the numerical approximation of $S(t_k, (i - \frac{1}{2})\Delta x, (j - \frac{1}{2})\Delta x)$ etc. Moreover, we denote the (variable) time step size $\tau^k := t_{k+1} - t_k$.

In order to be able to write the discretised equations in standard matrix-vector notation, we introduce the lexicographical grid ordering $\pi(i, j) = (i - 1)n_1 + j$. The vectors $\mathcal{S}, \mathcal{P}, \mathcal{X}, \mathcal{Y}$ are then defined by their coefficients as $\mathcal{X}_p = X_{i,j}$ with $p = \pi(i, j)$, etc. For the discretization of the diffusion operator we introduce for every grid point the grid neighborhood \mathcal{N}_p as the index set of direct neighbor points. For cell centers (i, j) in the interior of the domain $1 < i < n_1, 1 < j < n_2$ this is the set with four elements $\mathcal{N}_{p(i,j)} = \{p(i \pm 1, j), p(i, j \pm 1)\}$. In the case of grid cells that are aligned with boundaries of the domain, i.e., $i = 1$ or $i = n_1$ or $j = 1$ or $j = n_2$, this definition of the grid neighborhood leads to ghost cells outside the domain. These can be eliminated in the usual way by substituting the boundary conditions. The discretized equations read

$$\begin{aligned} \mathcal{S}_p^{k+1} = & \mathcal{S}_p^k - \tau^k \left(\frac{\mu_1}{\Upsilon_1} \frac{\mathcal{S}_p^k}{\kappa_1 + \mathcal{S}_p^k} \mathcal{X}_p^k + \frac{\mu_2}{\Upsilon_3} \frac{\mathcal{S}_p^k}{\kappa_2 + \mathcal{S}_p^k} \frac{\mathcal{P}_p^k}{\kappa_3 + \mathcal{P}_p^k} \mathcal{Y}_p^k \right) + \\ & + \frac{\tau^k}{2\Delta x^2} \sum_{s \in \mathcal{N}_p} (d_1(\mathcal{M}_s^k) + d_1(\mathcal{M}_p^k)) (\mathcal{S}_s^{k+1} - \mathcal{S}_p^{k+1}) \end{aligned} \quad (7)$$

-convection

$$\begin{aligned} \mathcal{P}_p^{k+1} = & \mathcal{P}_p^k - \tau^k \frac{\mu_2}{\Upsilon_2} \frac{\mathcal{S}_p^k}{\kappa_2 + \mathcal{S}_p^k} \frac{\mathcal{P}_p^k}{\kappa_3 + \mathcal{P}_p^k} \mathcal{Y}_p^k + \\ & + \frac{\tau^k}{2\Delta x^2} \sum_{s \in \mathcal{N}_p} (d_2(\mathcal{M}_s^k) + d_2(\mathcal{M}_p^k)) (\mathcal{P}_s^{k+1} - \mathcal{P}_p^{k+1}) \end{aligned} \quad (8)$$

-convection

$$\begin{aligned} \mathcal{X}_p^{k+1} = & \mathcal{X}_p^k + \tau^k \left(\mu_1 \frac{\mathcal{S}_p^k}{\kappa_1 + \mathcal{S}_p^k} - k_1 \right) \mathcal{X}_p^{k+1} + \\ & + \frac{\tau^k}{2\Delta x^2} \sum_{s \in \mathcal{N}_p} (D(\mathcal{M}_s^k) + D(\mathcal{M}_p^k)) (\mathcal{X}_s^{k+1} - \mathcal{X}_p^{k+1}) \end{aligned} \quad (9)$$

$$\begin{aligned} \mathcal{Y}_p^{k+1} = & \mathcal{Y}_p^k + \tau^k \left(\mu_2 \frac{\mathcal{S}_p^k}{\kappa_2 + \mathcal{S}_p^k} \frac{\mathcal{P}_p^k}{\kappa_3 + \mathcal{P}_p^k} - k_2 \right) \mathcal{Y}_p^{k+1} + \\ & + \frac{\tau^k}{2\Delta x^2} \sum_{s \in \mathcal{N}_p} (D(\mathcal{M}_s^k) + D(\mathcal{M}_p^k)) (\mathcal{Y}_s^{k+1} - \mathcal{Y}_p^{k+1}) \end{aligned} \quad (10)$$

In (7) and (8) any standard scheme from the Finite Volume Method literature can be used to discretize the convective contribution, e.g., those in [18]. In complying with the rule of constructing nonstandard-finite difference schemes to keep the order of the discrete difference approximation the same as the order of the continuous derivative [17], we use a simple 1st order upwinding method. For non-homogeneous Dirichlet or Neumann boundary cells, additional boundary contributions are obtained on the right hand side in the usual manner.

The new flow variables U^{k+1} are computed after the new values for \mathcal{X}, \mathcal{Y} are available, i.e., for the new biofilm structure. We use the analytical solution to (4) to approximate the flow velocities on the same uniform grid that was used for the other variables, albeit in a staggered fashion. The pressure is evaluated in the cell centers like the dissolved and the particulate substrates. The flow velocity components are evaluated on the cell edges, i.e., $u_{i,j}$ approximates the primary flow velocity in the point $(i\Delta x, (j - \frac{1}{2})\Delta x)$, and $v_{i,j}$ approximates the secondary flow velocity in $((i - \frac{1}{2})\Delta x, j\Delta x)$. After rearranging terms, equations (7)-(10) can be written as five linear systems that need to be solved subsequently in every time-step.

The numerical discretization method is standard for the non-degenerate equations describing the dissolved substrates S and P . More interesting is the computation of the biomass fractions X, Y because of the degenerate and fast diffusion

effects. In [6] it was shown that this discretization method applied to a single species biofilm model renders the following properties of the solution of the underlying continuous model: (i) positivity, (ii) boundedness by 1, (iii) finite speed of interface propagation, (iv) a discrete interface condition that corresponds to the continuous interface condition of the PDE, (v) a sharp interface in the numerical solution with only weak interface smearing effects, (vi) monotonicity of solutions near the interface and well-posedness of the merging of colonies (no spurious oscillations).

Proposition 1. *These properties carry over to the multi-species biofilm system investigated here.*

Proof. To see this we re-write (9)-(10) in matrix-vector form

$$\begin{cases} (\mathcal{I} - \tau^k \mathcal{D}^k - \tau^k \mathcal{G}_X^k) \mathcal{X}^{k+1} = \mathcal{X}^k \\ (\mathcal{I} - \tau^k \mathcal{D}^k - \tau^k \mathcal{G}_Y^k) \mathcal{Y}^{k+1} = \mathcal{Y}^k \end{cases} \quad (11)$$

where \mathcal{I} is the $n_1 n_2 \times n_1 n_2$ identity matrix, the matrix \mathcal{D}^k contains the diffusive contribution, i.e. $(\mathcal{D}^k)_{p,p} = -\frac{1}{2\Delta x^2} \sum_{s \in \mathcal{N}_p} (D(\mathcal{M}_s^k) + D(\mathcal{M}_p^k))$ and $(\mathcal{D}^k)_{p,s} = \frac{1}{2\Delta x^2} (D(\mathcal{M}_s^k) + D(\mathcal{M}_p^k))$ for component p and $s \in \mathcal{N}_p$, and the diagonal matrices $\mathcal{G}_{X,Y}^k$ contain the reaction terms $\mathcal{G}_X^k = \text{diag}_{p=1,\dots,n_1 n_2} (\mu_1 \mathcal{S}_p^k / (\kappa_1 + \mathcal{S}_p^k) - k_1)$, $\mathcal{G}_Y^k = \text{diag}_{p=1,\dots,n_1 n_2} (\mu_2 \mathcal{S}_p^k \mathcal{P}_p^k / [(\kappa_2 + \mathcal{S}_p^k)(\kappa_3 + \mathcal{P}_p^k)] - k_2)$.

The positivity of the biomass fractions \mathcal{X}, \mathcal{Y} follows with the same argument that was used in [6] for the single species model: the matrix \mathcal{D}^k is (at least) weakly diagonal dominant with negative diagonal entries and positive off-diagonal entries. If τ^k is chosen such that the diagonal matrices $\mathcal{I} - \tau^k \mathcal{G}_X^k$ and $\mathcal{I} - \tau^k \mathcal{G}_Y^k$ are positive, then the system matrices in (11) are diagonally dominant with positive diagonal entries and negative off-diagonal entries. Thus, they are M -matrices [11] and, therefore, their inverses are non-negative. Hence, if $\mathcal{X}^k, \mathcal{Y}^k$ are non-negative, then so are $\mathcal{X}^{k+1} = (\mathcal{I} - \tau^k \mathcal{D}^k - \tau^k \mathcal{G}_X^k)^{-1} \mathcal{X}^k$ and $\mathcal{Y}^{k+1} = (\mathcal{I} - \tau^k \mathcal{D}^k - \tau^k \mathcal{G}_Y^k)^{-1} \mathcal{Y}^k$. The sufficient time-step restriction for τ^k depends on the growth rates only. In fact τ^k is bounded by the reciprocal of the maximum growth rate, $\max\{\mu_1, \mu_2\}$. This is the time-scale for growth. Since we want to describe the time-evolution of the biofilm, i.e., resolve a time-scale not bigger than the time-scale for growth, the restriction placed here on the time-step is not critical for practical purposes.

Having positivity of \mathcal{X} and \mathcal{Y} established, we can deduce the remaining properties with the following trick: adding equations (9), (10), we obtain for $\mathcal{M} := \mathcal{X} + \mathcal{Y}$ the (component wise) inequality

$$\begin{aligned} \mathcal{M}^k &= (\mathcal{I} - \tau^k \mathcal{D}^k) \mathcal{M}^{k+1} - \tau^k [\mathcal{G}_X^k \mathcal{X}^{k+1} + \mathcal{G}_Y^k \mathcal{Y}^{k+1}] \geq \\ &\geq (\mathcal{I} - \tau^k \mathcal{D}^k - \tau^k [\mu \mathcal{S}_p^k / (\tilde{\kappa} + \mathcal{S}_p^k) - \kappa]) \mathcal{M}^{k+1} \end{aligned} \quad (12)$$

where $\mu := \max\{\mu_1, \mu_2\}$, $\tilde{\kappa} := \max\{\kappa_1, \kappa_2\}$ and $\kappa := \min\{k_1, k_2\}$. We denote by $\tilde{\mathcal{M}}^{k+1}$ the solution of the associated linear system

$$(\mathcal{I} - \tau^k \mathcal{D}^k - \tau^k [\mu \mathcal{S}_p^k / (\tilde{\kappa} + \mathcal{S}_p^k) - \kappa]) \tilde{\mathcal{M}}^{k+1} = \mathcal{M}^k. \quad (13)$$

The matrix here is again an M -matrix, i.e. its inverse has only non-negative entries. Then it follows from (12) and (13) immediately that $\tilde{\mathcal{M}}^{k+1} \geq \mathcal{M}^{k+1}$. Moreover, as a solution of (12), $\tilde{\mathcal{M}}$ is also a numerical solution of the single species biofilm equation studied in [6]. Thus, it has the properties (i)-(vi) above. It follows, again due to positivity, immediately that also \mathcal{X}^{k+1} and \mathcal{Y}^{k+1} are bounded by 1, have a finite speed of interface propagation, show only weak interface smearing effects and are oscillation free.

The linear systems in (7)-(10) are at least weakly diagonal dominant, sparse and structurally symmetric but not symmetric. We store them in diagonal format. Among the class of Krylov subspace methods the *stabilised bi-conjugated gradient method* (cf. Alg. 1, [22]) is a proper choice for their solution. It requires in every iteration two matrix-vector products, 4 inner products with reduction and a number of vector additions and scalar-vector multiplications, which can be easily parallelized on the loop-level.

Algorithm 1. *The stabilized bi-conjugate gradient method for the linear system $Ax = b$. Greek variables denote real scalars, lower case Latin variables are real vectors [22].*

- [0] $r := b - Ax, r_0 := r, p := r, \rho := r_0^T r$
- [i] $v := Ap$
- [ii] $\alpha := \rho / r_0^T v$
- [iii] $s := r - \alpha v$
- [iv] $t := As$
- [v] $\omega := t^T s / t^T t$
- [vi] $x := x + \alpha p + \omega s$
- [vii] $r := s - \omega t$
- [viii] $\beta := \rho, \rho := r_0^T r, \beta = \rho\alpha/(\beta\omega), p := r + \beta(p - \omega v), \text{ continue at [i]}$

3 Computational Simulation

3.1 Computational Setup

We simulate model (1) in a rectangular domain of $5 \times 0.5\text{mm}^2$, discretised by 2000×200 grid cells. The code is written in Fortran 95. For the solution of the sparse system the recursive communication based Fortran 77 source code library SPARSKIT [22] is used after OpenMP parallelization of vector operations in the BiCGSTAB algorithm Alg. 1, the matrix-vector product routine is provided externally. Also the calculation of the nonlinear reaction terms in (7)-(10) is carried out in parallel.

The following computers were used in our study: (a) a SGI Altix 330 with 16 Itanium II processors at 1.5GHz, (b) a SGI Altix 350 with 32 Itanium II processors at 1.6GHz, (c) a SGI Altix 450 with 32 dual core Itanium II processors at 1.6GHz, (d) a SGI ALTIX 3700 with 128 Itanium II processors at 1.6GHz, and (e), (f) two commodity-built workstations with two quadcore Xeon processors each at 3.0 GHz. All machines run under SUSE Linux. The Intel Fortran Compiler was used for compilation. Machines (a), (b), (d) are of the same generation,

but (d) is of a higher end class then its smaller cousins (a), (b). Machine (c) is one generation newer than these machines and practically the successor model of (b).

3.2 A Typical Simulation

A typical simulation of model (I) is visualised in Figures 1 and 2 for four selected time steps (snapshots). The parameters used in this simulation are summarised in Table 2.

Shown are the substrate concentration S in Figure 1 and the pollutant concentration P in Figure 2, color coded in greyscale. The substrate concentrations on inflow are prescribed at values clearly above the half saturation constants. The biofilm/water interface, i.e. the curve $\bar{\Omega}_1(t) \cap \bar{\Omega}_2(t)$, is plotted in white. Primary flow direction is from top to bottom at a low hydrodynamic regime, $Re = 0.0005$. Biofilm grows on one side of the flow channel from a dense random inoculation. Initially not enough biomass is in the system to degrade dissolved substrates notably. The biomass grows at almost maximum growth rate.

Table 2. Model parameters used in this simulation study. Typical biofilm growth parameters are taken from [24], parameters for pollution degradation were adapted from [3], biofilm spreading and hydrodynamic parameters from [9].

parameter	symbol	value	unit
length of flow channel	L	0.005	m
height of flow channel	H	0.0005	m
growth rate of biofilm stabiliser	μ_1	6	1/d
growth rate of pollutant degrader	μ_2	3	1/d
half saturation constants:			
biofilm stabilizer w.r.t. nutrient	κ_1	4	g/m ³
pollutant degrader w.r.t. nutrient	κ_2	0.4	g/m ³
pollutant degrader w.r.t. pollutant	κ_3	0.4	g/m ³
decay rate of biofilm stabilizer	k_1	0.030	1/d
decay rate of pollutant degrader	k_2	0.015	1/d
substrate utilization			
biofilm stabilizer w.r.t. nutrient	γ_1	10^{-3}	m ³ /g
pollutant degrader w.r.t. nutrient	γ_2	$1.25 \cdot 10^{-6}$	m ³ /g
pollutant degrader w.r.t. pollutant	γ_3	$2.5 \cdot 10^{-6}$	m ³ /g
diffusion coefficients:			
nutrient in water	$d_1(0)$	$2 \cdot 10^{-4}$	m ² /d
nutrient in biofilm	$d_1(1)$	$1.9 \cdot 10^{-4}$	m ² /d
pollutant in water	$d_2(0)$	$1 \cdot 10^{-4}$	m ² /d
pollutant in biofilm	$d_2(1)$	$0.8 \cdot 10^{-4}$	m ² /d
Reynolds number	Re	0.0005	–
nutrient bulk concentration	S_∞	20	g/m ³
pollutant bulk concentration	P_∞	20	g/m ³
biofilm mobility parameter	δ	$2 \cdot 10^{-12}$	m ² /d
biofilm mobility exponent	a	4	–
biofilm mobility exponent	b	4	–

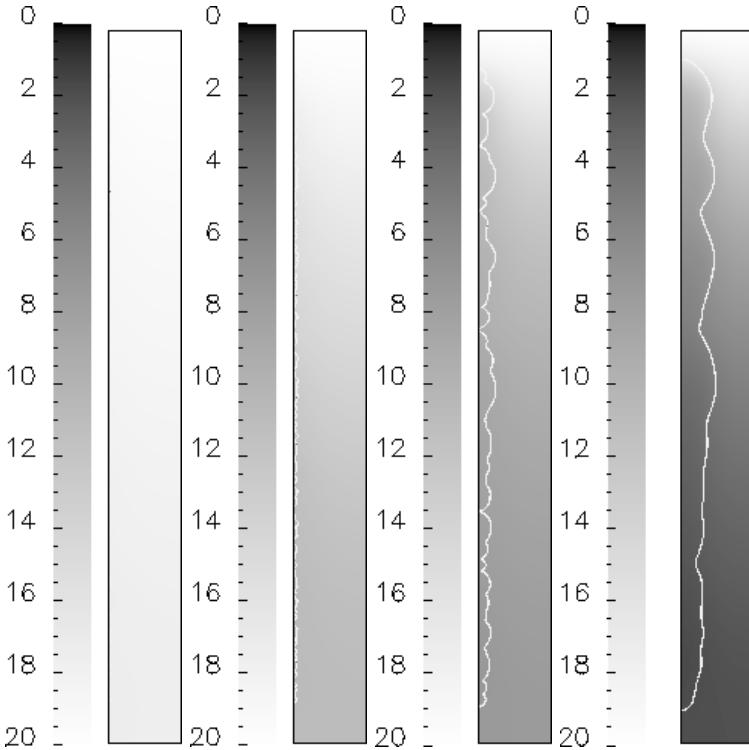


Fig. 1. Nutrient concentration S for $t = 0.25, 0.75, 1.25, 1.5d$; the white line indicates the biofilm/water interface $\bar{\Omega}_1(t) \cap \bar{\Omega}_2(t)$

Eventually, the amount of active biomass in the system becomes large enough to inflict notable substrate degradation. Substrate gradients are observed in flow direction and from the biofilm toward the liquid phase. While a clear nutrient degradation is observed, S does not drop notably below the half saturation value, indicating that bacteria are not experiencing severe starvation, cf. Fig 1. On the other hand, pollutant becomes strongly limited everywhere in the biofilm and in the bulk in the downstream in the third time step shown, Fig 2. This indicates that the pollutant degrader is growth limited. Together with the observation made for the nutrient above, this implies that for larger t biofilm growth is primarily due to growth of the biofilm stabilizer X . This is also verified in Figure 3, where the total amount of biomass of both species in the system is plotted, i.e.

$$X_{total}(t) = \int_{\Omega} X(t; x) dx, \quad Y_{total}(t) = \int_{\Omega} Y(t; x) dx.$$

These functions are plotted in log-scale. As long as log-scale plots are straight lines, the biomass grows at maximum growth rate, i.e. $S/(\kappa_1 + S) \approx 1$ everywhere and the growth curve behaves essentially like a linear growth model

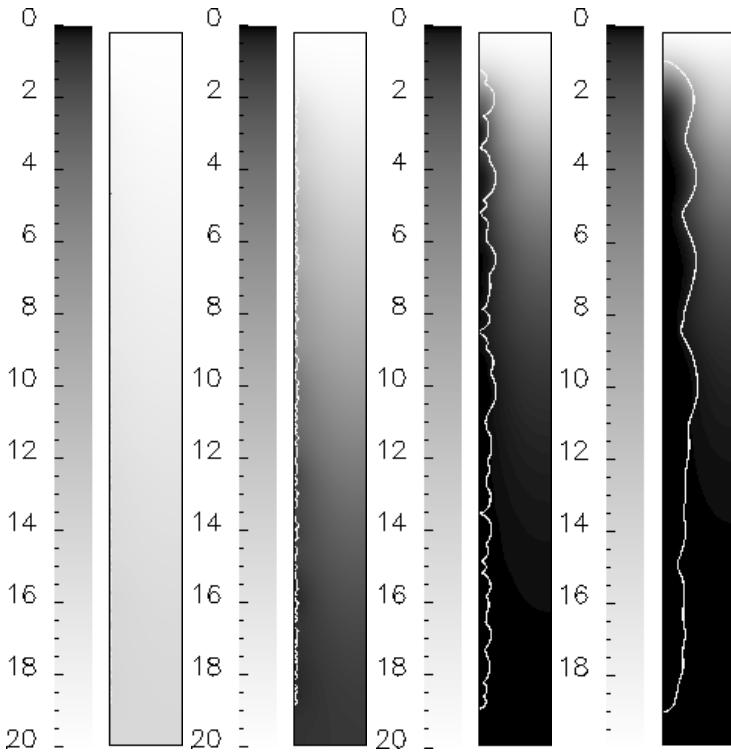


Fig. 2. Pollutant concentration P for $t = 0.25, 0.75, 1.25, 1.5d$; the white line indicates the biofilm/water interface $\bar{\Omega}_1(t) \cap \bar{\Omega}_2(t)$

$\dot{X}_{total} \approx (\mu_1 - k_1)X_{total}$, thus $X_{total}(t) \approx X_{total}(0)e^{(\mu_1 - k_1)t}$ (similar for Y). Growth limitations due to substrate limitations are indicated by the growth curve remaining below the line of maximum growth.

3.3 Parallel Performance

It has been reported in benchmarking studies that execution and overhead time variations are seemingly inherent in OpenMP programs, if the same task is completed several times, cf. [20][21]. While the magnitude of these variations appears to change across platforms, it seems that they always exist. An explanation for these variations has not be given, but it is known that in addition to parallelisation issues (synchronisation, scheduling etc), modern sophisticated processors are susceptible to intermittent and temporary hardware failure [12][14][25]. Usually benchmarking efforts center on relatively simple tasks, such as communication primitives, basic vector operations etc. For all practical purposes it is not obvious how the reported variations in execution time carry over to more complex and composite tasks like the biofilm simulations at hand, i.e., whether they amplify

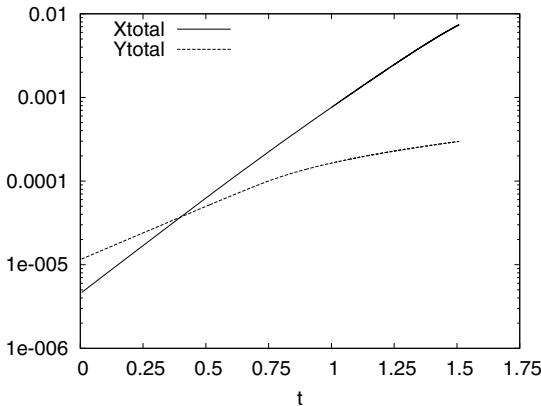


Fig. 3. Total volume fraction (in logscale) occupied by biofilm stabilizer X and pollutant degrader Y over the course of the simulation

or cancel each other (dampen). Therefore, in order to investigate parallel gain of the code, we solve on every machine and for a given number of processors the simulation problem five times and calculate from the execution times the average and the standard deviation, cf. Figure 4

Parallel acceleration is achieved on all computers tested. The Xeon based machines are slowest. The Itanium based machines are ordered as expected, i.e., the newest machine (Altix 450) is fastest. The rather similar Altix 330 and 350 of the previous generation of this type are almost equally fast; the Altix 3700 which is of the same generation as the 330 and the 350 but of a higher quality family is almost as fast as the newer Altix 450 and faster than its smaller cousins of the same generation.

On the Altix 330 with 16 single core Itanium processors (a), computing time decreases with increasing number of processors used, all the way through to 16. On the Altix 350 with 32 single core Itanium processors (b), speed-up saturation is reached at 8 processors. On the Altix 450 with 32 dual core Itanium processors (c), computing time decreases up to 16 cores but not beyond that. On the commodity built workstations (e), (f) with two quadcore Xeon processors performance increases up to 5 cores but not beyond that. This indicates, however, that both architectural features, multi-core and multi-processor, enable acceleration. On the Altix 3700 with 128 Itanium processors (d), an overall speed-up trend is observed, but computing time is not a decreasing function of the number of processors. The most likely explanation for this behavior is the usage pattern of this machine. The simulations on all machines were carried out during regular operation. The Altix 3700 is operated by SHARCNET in a batch environment and most of the time under heavy use by several users at a time. On all other platforms, while no special provisions were made, the machines were for the most part fully available for our purposes.

The smallest variations in computing time were observed on the newer Altix 450 (c), and there only for the smallest tested number of two cores. If more cores

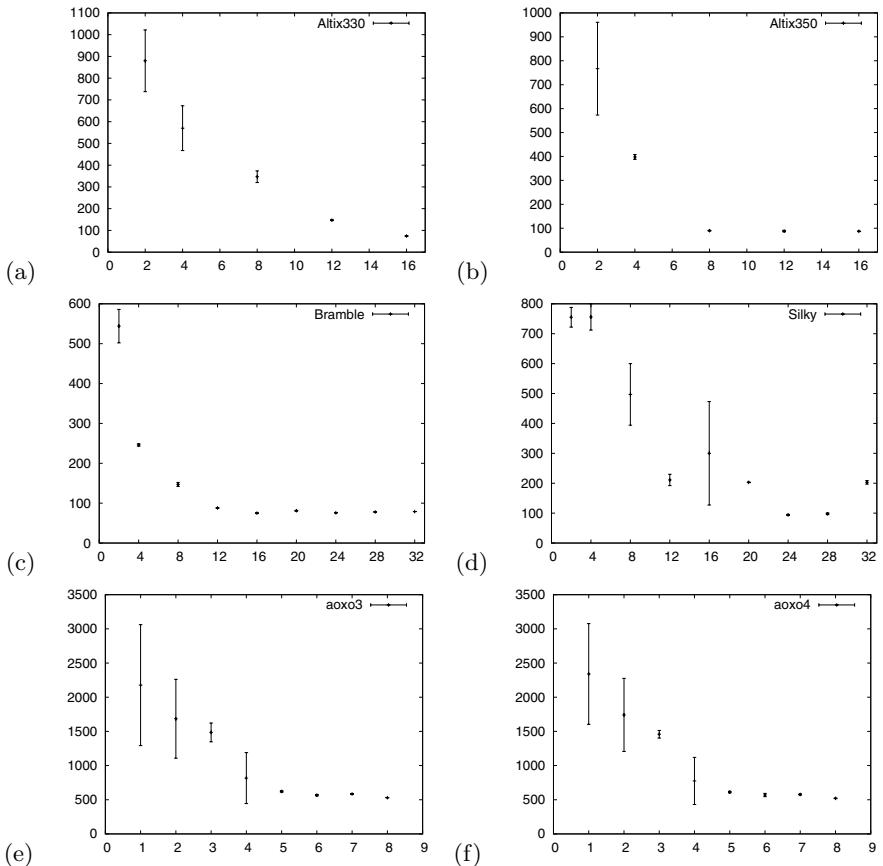


Fig. 4. Average computing time (in minutes) on different number of processors/cores, with standard error bars. Here, X-axis and Y-axis represent number of processors and time in minutes respectively. (a) Altix 330 (b) Altix 350 (c) Altix 450 (d) Altix 3700, (e), (f) commodity built desktop server.

were used in the simulation, execution times were rather stable. A qualitatively similar picture was obtained for the Altix 350, although the span of variations was larger for two processors than for the Altix 450. Also on the Altix 330, the variation decreases with increasing number of processors, although they practically disappear only at 12 or more processors. An entirely different picture was obtained again for the Altix 3700, where the pattern of variations was quite irregular. Again, this is most likely attributed to the usage of the machine. On the commodity built workstation, the variation pattern is largely similar to the smaller Altix machines, i.e., variations disappear for large enough number of cores. However, on both machines, an exception is the case of three cores, where the variation is smaller than for two or four cores.

All in all, for our OpenMP application the simpler desktop workstations with two standard PC market quadcore Xeon processors behave qualitatively similar as the native parallel computers. Of course there are quantitative differences. For one, the faster technology of the Altix systems allows a faster computation. Secondly, the limitations for parallel speedup are reached at a lower number of cores. Nevertheless, the results imply that the simple desktop workstation can be reliably used for development of parallel applications in Computational Science and Engineering for the higher performing compute servers.

4 Conclusion

We presented a Mickens scheme for a highly nonlinear system of parabolic evolution equations. The scheme is constructed such that the problem is essentially reduced to solving sparse, well behaved linear systems and to evaluate nonlinear highly-local reaction terms. Thus, it is straightforward to parallelize in OpenMP, which allows us to make use of multi-core/multi-processor architectures. We tested the simulation code on Itanium based SGI Altix systems as well as on commodity built Xeon based desktop PCs. On both types of platform we found that parallel execution times are subject to variations. Such variations have been observed in benchmarking studies of computational primitives before. Our results confirm that these fluctuations do not necessarily cancel out in more complex simulation tasks that involve myriads of primitive and basic parallel operations, in particular for small thread numbers. For larger thread numbers on most machines tested the variations became much smaller. The biggest fluctuations in computing times were observed at the commodity built computers and on the largest machine, which is used in a batch environment and generally in heavy use. It follows that actual computing times are not always easy to predict. While OpenMP is certainly a convenient and straightforward approach to use, such strong fluctuations might be considered unsatisfactory for some applications and in some environments. We saw that cheaper commodity built desktop computers behave qualitatively quite similar as more expensive mid-range parallel compute servers, with respect to their parallelization behavior. In other words they allow to take the same advantage of OpenMP parallelization. Of course, there are quantitative differences due to the differences in technology used in these machines.

In the introduction we pointed out that there are essentially two approaches to deal with the complexity of computational problems of the type considered here on parallel computers: (i) run several independent serial simulations concurrently on individual processors [i.e., a serial farm approach], or (ii) run the simulation in parallel and distributed over several processors and cores. Our study addressed the approach (ii). However, our results showed that, depending on hardware used, parallel speedup might reach its limitations for core/processor numbers smaller than the maximum number available, e.g., on the Altix 450. This permits to run several parallel simulation concurrently which can greatly reduce the overall computing time of an extensive numerical experiments that can consist of dozens of runs, i.e. a computation of the approaches (i) and (ii).

Despite the mathematical peculiarities of our model, we consider our simulation problem, as a convection-diffusion-reaction system, a rather typical problem in computational science and engineering. Therefore, we expect that the findings of our case study will have validity beyond our specific investigation.

Acknowledgment

The hardware used in this study was provided by (a) a NSERC *Research Tools and Infrastructure I* grant (PI: HJE), (b) a Canada Foundation for Innovation *New Opportunities Grant* with matching from the Ontario Ministry of Research and Innovation (PI: M. Cojocaru, Dept. Math and Stats, Guelph), (c), (e), (f) a Canada Foundation for Innovation *Leaders Opportunity Grant (CRC support)* with matching funds from the Ontario Ministry of Research and Innovation (PI: HJE), and (d) the Shared Hierarchic Academic Research Computing Network (SHARCNET). We thank SHARCNET's Technical Staff (Kaizaad Bilmorya, Doug Roberts) for the administration of (c), (d) and departmental IT staff (Larry Banks) for the administration of (a), (b), (e), (f). This project was supported by the Canada Research Chairs Program and the Discovery Grant program of the Natural Science and Engineering Research Council with grants awarded to HJE.

References

- Anguelov, R., Lubuma, J.M.S.: Contributions to the mathematics of the nonstandard finite difference method and its applications. *Num. Meth. PDE* 17, 518–543 (2001)
- Bull, J.M.: Measuring synchronisation and scheduling overheads in OpenMP. In: Proc. 1st Europ. Workshop on OpenMP, EWOMP, Lund (1999)
- Chen-Carpentier, B.M., Kojouharov, H.V.: Numerical simulation of dual-species biofilms in porous media. *Appl. Num. Math.* 47, 377–389 (2003)
- Curtis-Maury, M., Ding, X., Antonopoulos, C.D., Nikolopoulos, D.S.: An evaluation of OpenMP on current and emerging multithread/multicore processors. In: Mueller, M.S., Chapman, B.M., de Supinski, B.R., Malony, A.D., Voss, M. (eds.) IWOMP 2005 and IWOMP 2006. LNCS, vol. 4315, pp. 133–144. Springer, Heidelberg (2008)
- Demaret, L., Eberl, H., Efendiev, M., Lasser, R.: Analysis and simulation of a meso-scale model of diffusive resistance of bacterial biofilms to penetration of antibiotics. *Adv. Math. Sc. and Appls.* 18(2), 269–304 (2008)
- Eberl, H.J., Demaret, L.: A finite difference scheme for a doubly degenerate diffusion-reaction equation arising in microbial ecology, *El. J. Diff. Equ. CS* 15, 77–95 (2007)
- Eberl, H.J., Muhammad, N., Sudarsan, R.: Computing Intensive Simulations in Biofilm Modeling. In: Proc. 22nd Int. High Performance Computing Systems and Applications (HPCS 2008), (IEEE Proceedings), Quebec City, pp. 132–138 (2008)
- Eberl, H., Parker, D., van Loosdrecht, M.: A new deterministic spatio-temporal continuum model for biofilm development. *J. Theor. Medicine* 3(3), 161–176 (2001)

9. Eberl, H.J., Sudarsan, R.: Exposure of biofilms to slow flow fields: the convective contribution to growth and disinfections. *J. Theor. Biol.* 253(4), 788–807 (2008)
10. Efendiev, M.A., Zelik, S.V., Eberl, H.J.: Existence and longtime behavior of a biofilm model. *Comm. Pure Appl. Analysis* 8(2), 509–531 (2009)
11. Hackbusch, W.: Theorie und Numerik elliptischer Differentialgleichungen. Teubner, Stuttgart (1986)
12. Herbert, S., Marculescu, D.: Characterizing chip-multiprocessor variability-tolerance. In: Proc. 45th Ann. Conf. Design Automation, Anaheim, pp. 313–318. ACM Digital Library, New York (2008)
13. Herlihy, M., Luchangco, V.: Distributed computing and the multicore revolution. *ACM SIGCAT News* 39(1), 62–72 (2008)
14. Humenay, E., Tarjan, D., Skadron, K.: Impact of process variations on multicore performance symmetry. In: Proc. Conf. Design Automation and Test in Europe, DATE 2007, pp. 1653–1658. ACM Digital Library, New York (2007)
15. Liao, C., Liu, Z., Huang, L.L., Chapman, B.: Evaluating openMP on chip multi-Threading platforms. In: Mueller, M.S., Chapman, B.M., de Supinski, B.R., Malony, A.D., Voss, M. (eds.) IWOMP 2005 and IWOMP 2006. LNCS, vol. 4315, pp. 178–190. Springer, Heidelberg (2008)
16. Marowka, A.: Parallel computing on any desktop. *Comm. ACM* 50(9), 75–78 (2007)
17. Mickens, R.E.: Nonstandard finite difference schemes. In: Mickens, R.E. (ed.) Applications of nonstandard finite difference schemes. World Scientific, Singapore (2000)
18. Morton, K.W.: Numerical solution of convection-diffusion problems. Chapman & Hall, London, Boca Raton (1996)
19. Pankratius, V., Schaefer, C., Jannesari, A., Tichy, W.F.: Software engineering for multicore systems - an experience report. In: Proc. 1st Int. Workshop Multicore Software Engineering, Leipzig, pp. 53–60. ACM Digital Library, New York (2008)
20. Reid, F.J.L., Bull, J.M.: OpenMP Microbenchmarks Version 2.0. HPCx Technical Report, HPCxTR0411, 28 p. (2004)
21. Reid, F.J.L., Bull, J.M.: OpenMP Microbenchmarks Version 2.0. In: Proc. EWOMP 2004, pp. 63–68 (2004)
22. Saad, Y.: SPARSKIT: a basic tool kit for sparse matrix computations (1994), <http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html>
23. Thomaszewski, B., Pabst, S., Blochinger, W.: Parallelism in physically-based simulations on multi-core processor architectures. *Computers & Graphics* 32(1), 25–40 (2008)
24. Wanner, O., Eberl, H., Morgenorth, O., Noguera, D., Picioreanu, D., Rittmann, B., van Loosdrecht, M.: Mathematical Modelling of Biofilms. IWA Publishing, London (2006)
25. Wells, P.M., Chakraborty, K., Sohi, G.S.: Adapting to Intermittent Faults in Multi-core Systems. In: Proc. ASPLOS 2008, Seattle, pp. 255–264. ACM Digital Library, New York (2008)

An Evaluation of Parallel Numerical Hessian Calculations

Mark S. Staveley, Raymond A. Poirier, and Sharene D. Bungay

Department of Computer Science
and Department of Chemistry
Memorial University of Newfoundland,
St. John's, NL, A1B 3X5, Canada
{mark.staveley,rpoirier,sharene}@mun.ca
<http://www.mun.ca>

Abstract. The calculation of the Hessian (or force constant matrix), is an important part of geometry optimization calculations. The process of calculating the Hessian can be demanding in both computational time and resources. Even though many computational chemistry software packages make use of numerical Hessian calculations, very little has been published with respect to the performance and accuracy traits of this type of calculation. This paper presents the following: a comparison of different parallelization methods as they are applied to the numerical Hessian calculation; the accuracy of the results produced by these different parallelization methods; and an evaluation that compares the performance of the numerical Hessian parallel algorithm on different system configurations.

Keywords: Computational Chemistry, Parallelization, Numerical Hessian, Geometry Optimization Calculations.

1 Introduction

Taking advantage of parallel computing environments is becoming increasingly important as scientific calculations continue to increase in size and complexity. The need for parallel algorithms is amplified in the area of quantum chemistry as many computational methods are severely limited by the computer resources that are typically available within a single machine. By employing algorithms that utilize the hardware components of many machines simultaneously, we are able to remove some of the obstacles that have been limiting the size and type of problems that can be solved.

Typically, finding and characterizing stationary points on potential energy surfaces of reactions and chemical systems involves three steps. The first step involves the solving of either the Schrödinger equation or the Kohn-Sham equation. The second step is the process of determining the first derivative, otherwise

known as the gradient. The final step is the evaluation of the Hessian (1; 2). The Hessian is the second derivative matrix of the total energy with respect to the co-ordinates (either internal or Cartesian). When calculated at a stationary point, the Hessian corresponds to the force constant matrix, which can be used to calculate vibrational frequencies.

The Hessian can be calculated either analytically or numerically. The main advantage of the analytical method is that it is more accurate than the numerical method. However, the analytical second derivative must be programmed and available for each level of theory. As a result, the software to calculate second derivatives analytically using higher levels of theory, which are more complex, may not be available. The numerical method, on the other hand, uses a finite difference and only requires the first derivative. This method is easier to implement, and unlike the analytical Hessian, has the ability to restrict calculations to a particular area of interest.

A common calculation that is performed using electronic structure theory is a geometry optimization. The geometry optimization process determines a stationary point for a given chemical structure. The first part of the geometry optimization process involves the calculation of a function evaluation (SCF + gradient), and the calculation or approximation of the Hessian. Subsequent steps simply perform a Hessian update rather than perform a recalculation. If we assume that an SCF calculation takes time t , then the gradient will take approximately $1.5t$, and the Hessian calculation (when computed analytically) will take approximately $10(t + 1.5t)$. This would give a total time of approximately $27.5t$, with 91%(25t) of the computational work being done attributed to the analytical Hessian calculation. In comparison, when computing the Hessian using a numerical method, the time depends on the number of optimization parameters (equal to $3n$, where n is the number of atoms within the structure). Following the above example, if one is using Cartesian coordinates and has 48 optimization parameters, then the numerical Hessian (using a central difference method, for which two function evaluations are required) would take $(2 \times 48)(t + 1.5t)$ plus the time $2.5t$ for the initial function evaluation, giving a total time of $242.5t$. In a calculation of this size, the numerical Hessian represents $240t/242.5t = 99\%$ of the computational work being done. However, when parallelizing the numerical Hessian, the work becomes distributed amongst many processors and as such the elapsed time of the calculation (or time to solution) decreases.

Many different computational chemistry codes allow for the calculation of the numerical Hessian in parallel. In this paper, we present a numerical Hessian algorithm that is both generic in nature and system independent. It is considered to be generic in nature as there are no components of the algorithm designed to optimize the numerical Hessian calculation itself (for example taking advantage of the symmetry found within a chemical structure), and it is considered to be system independent as only standard installations of Fortran 90 (Sun Studio compiler) and MPI (MPICH) have been used. As such, the algorithm can be considered as a minimum performance benchmark when evaluating the methodology, algorithm and implementation of a system that calculates

numerical Hessians. In addition to presenting this algorithm, an evaluation, including a decomposition methods comparison, a speedup, efficiency, and accuracy comparison, and a hardware / operating system comparison, is presented and discussed.

2 Design

The process of parallelizing the Hessian calculation was built around the notion that the finite difference calculations associated with the evaluation of the numerical Hessian can be distributed simultaneously to different processors. These finite difference calculations can be done using a forward difference,

$$f'' \approx \frac{f'(x + h) - f'(x)}{h}, \quad (1)$$

a backward difference,

$$f'' \approx \frac{f'(x) - f'(x - h)}{h}, \quad (2)$$

or a central difference,

$$f'' \approx \frac{f'(x + h) - f'(x - h)}{2h}, \quad (3)$$

approach, where h represents the step size of the finite difference interval, f' is the first derivative, and f'' is the second derivative.

The parallelization of the numerical Hessian calculation is done through the distribution of the finite difference calculations. When determining the numerical Hessian at least one function evaluation is required for each optimization parameter. One function evaluation is required for each optimization parameter when using the forward or backward difference methods, Equations (1) and (2), and two function evaluations are required for each optimization parameter when using the central difference method, Equation (3).

When distributing the finite difference calculations amongst different processors, the order in which the calculations are completed becomes important. In some cases rounding errors can be introduced into the calculations. An example of how the ordering can be changed for a set of calculations can be seen in Figure 1. In this example the calculation $C + D = CD$ never occurs when the calculations are completed in serial. This type of ordering change normally does not impact results, but it is possible that reordering can cause rounding errors.

During the process of completing the finite difference calculations, information from the previous step (or iteration) of the calculation is used. Since it is possible for the ordering of steps to be changed with different decomposition methods (see Figure 2), it becomes important to evaluate more than one decomposition method. Here, we are using two decomposition methods, namely sequential decomposition and block decomposition.

For sequential decomposition, the calculations are distributed in turn to each processor. For block decomposition, the calculations are divided up into equally

Serial	Parallel (2 Processors)
$A + B + C + D = \text{Answer}$	$A + B + C + D = \text{Answer}$
Order of instructions:	Order of instructions:
Processor 1 $A + B = AB$	Processor 1 $A + B = AB$
Processor 1 $AB + C = ABC$	Processor 2 $C + D = CD$
Processor 1 $ABC + D = ABCD$	Processor 1 $AB + CD = ABCD$

Fig. 1. Example showing the change in ordering of calculations (parallel and serial)

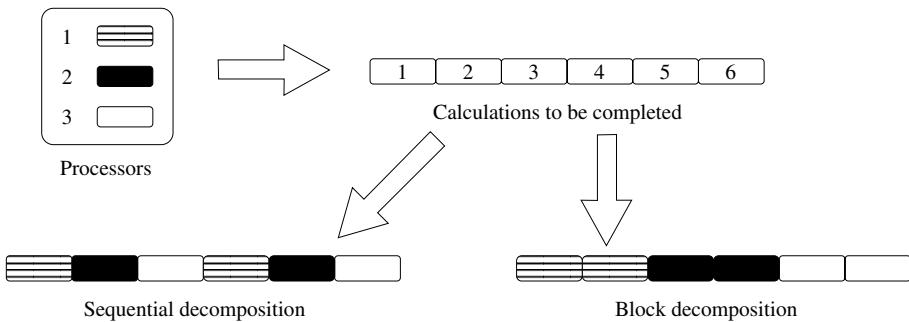


Fig. 2. Illustration of block and sequential decomposition techniques using 6 calculations and 3 processors

sized blocks, with each block then assigned to a processor. In the case of one processor, or when the number of processors is equal to the number of tasks, the sequential and block decomposition are equivalent. Figure 2 illustrates how the block and sequential methods operate on a six component problem when three processors are available.

3 Implementation

There are a number of ways that a developer can convert a parallel algorithm into a fully functioning piece of parallel code. Three of the most common parallelization methods are automatic parallelization, OpenMP, and the Message Passing Interface (MPI).

We chose to use MPI for our implementation. MPI is a library based parallel method that can be integrated with various compilers. Using MPI is more difficult than using OpenMP or automatic parallelization as the scope and context of data variables needs to be explicitly defined and subsequently either shared or kept private between working processes. Furthermore, the data that are used between processes must be sent and received using specialized message instructions to ensure that there are no complications such as race conditions, loss of data coherency, or deadlock. However, MPI does have the distinct advantage of

being able to span multiple machines, whereas both automatic parallelization and OpenMP are restricted to single-server images.

As part of our implementation, we used the MUNgauss quantum chemistry code (3) as a framework for implementing our algorithm in parallel. Our modified version of MUNgauss is able to perform forward, backward, and central difference numerical Hessian calculations in parallel, using either block or sequential decomposition. Again the authors would like to stress that although we have used MUNgauss in the implementation of our numerical Hessian algorithm, our algorithm is general enough that with the correct parallel constructs, any quantum chemistry code that only supports serial, or single-processor, numerical Hessian calculations could be easily parallelized.

4 Method

In order to fully evaluate our parallel algorithm, we considered speedup, parallel fraction, efficiency, and accuracy of our implementation.

Speedup was chosen as a method to make sure that we had targeted the correct areas in the code for parallelization. The speedup, S , of a program can be calculated as

$$S = \frac{T_{(1)}}{T_{(N)}}, \quad (4)$$

where $T_{(N)}$ represents the time required to complete the calculation using N processors, and $T_{(1)}$ represents the time required to complete the calculation using 1 processor. The parallel fraction, f , calculated as

$$f = \frac{1 - \frac{T_{(N)}}{T_{(1)}}}{1 - \frac{1}{N}}, \quad (5)$$

represents the fraction (or percentage) of the work that is being done in parallel. The efficiency, E , of a parallel program can be calculated using Amdahl's law (4),

$$E = \frac{T_{(1)}}{NT_{(N)}}, \quad (6)$$

and gives an indication of how well we are parallelizing our code.

For testing purposes, the central difference approach, Equation (3), was chosen because it produces results that are more accurate than the forward or backward difference approaches, Equations (1) and (2). However, the central difference method is more computationally intensive as the calculation of both the forward difference and the backward difference are required. Although the central difference method produces numerical Hessian results with the best possible accuracy (for a given basis set), the determination of these results requires approximately double ($2x$) the CPU resources than results determined using either the forward or backward difference methods.

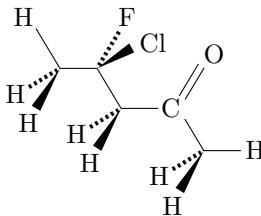


Fig. 3. Structure of C_5H_8ClFO

It is important to note that there are cases involving geometry optimization where the forward and backward difference methods will provide enough accuracy. However, for frequency calculations, the most accurate method is required to produce reasonable results. In order to evaluate trends in accuracy, speedup, and behaviour, several chemical structures were used for testing purposes.

Optimized geometries for CCl_4 , CBr_4 , and C_5H_8ClFO (see Figure 3) were used as the starting point for our calculations.

Cartesian coordinates and HF/6-31G(d) were used to compute all vibrational frequencies. Initial optimized geometries and analytical vibrational frequencies were produced using Gaussian03 (5). In order to evaluate the speedup of our implementation, different sets of processor configurations were used for all three of the test structures. CCl_4 and CBr_4 each required $30 + 1$ different gradient calculations, while C_5H_8ClFO required $96 + 1$ different gradient calculations. Both the sequential and block decomposition methods were tested with each of the three test structures. In all cases, the numerical Hessian calculations were distributed uniformly across processors. In the case of CCl_4 and CBr_4 , the calculations were distributed amongst 1, 3, 5, and 15 processors. For C_5H_8ClFO , the calculations were distributed amongst 1, 2, 3, 4, 6, 8, 12, 16, 24, and 48 processors.

Work done by Krishnan et al. (6) using NWChem (7) and (8), shows that the calculations associated with the numerical Hessian can be distributed amongst an architecture with standardized computer hardware components and interfaces (otherwise known as common component computer architecture). However, this investigation stopped after a performance increase, when compared to a single processor, was observed.

Our investigation is different in a number of ways. First, our algorithm is not specific to any particular type of hardware configuration or software package. In fact, we have attempted to make our algorithm as general as possible. Second, we not only include a performance (scaling) evaluation, but we also compare the performance when using different chemical structures. Third, we look at how well our implementation is parallelized, and how well this compares to the theoretical / predicted breakdown of parallel and serial work. Fourth, we evaluate the accuracy of our results by comparing numerical and analytical results. This accuracy evaluation is done by comparing frequencies that have been calculated using force constants obtained either numerically or analytically. Lastly,

we compare the performance obtained with different combinations of hardware and operating systems.

5 Results and Discussion

5.1 Performance

As part of the evaluation of our algorithm, we tested how well the representative MPI numerical Hessian implementation performed across multiple processors. Using our three test structures we compared the timings of the calculations while varying the number of processors. Figures 4, 5, and 6 illustrate the speedup and timing performance using both the block and sequential versions of our algorithm.

These results show that there are minimal differences in timing between the sequential and block decomposition methods for all of our test structures. CPU time was used as opposed to wall-clock time for all of our timing experiments.

Figures 4, 5, and 6 also show that our MPI implementation is scaling well but that we are not obtaining linear speedup in any of our test cases. Nonetheless, our results are quite an improvement when compared to the single CPU cases. In the case of CCl_4 the time decreased from 1073 seconds(s) to 119 s (9 times faster) when moving from a single processor core to 15 processor cores (see Table 1). We see a similar type of speedup when looking at CBr_4 where the time decreased from 2901 s to 318 s, again 9 times faster (see Table 2).

When looking at a more complex structure ($\text{C}_5\text{H}_8\text{ClFO}$) we see a speedup of more than 9 times with 12 processor cores. As we increase to the maximum

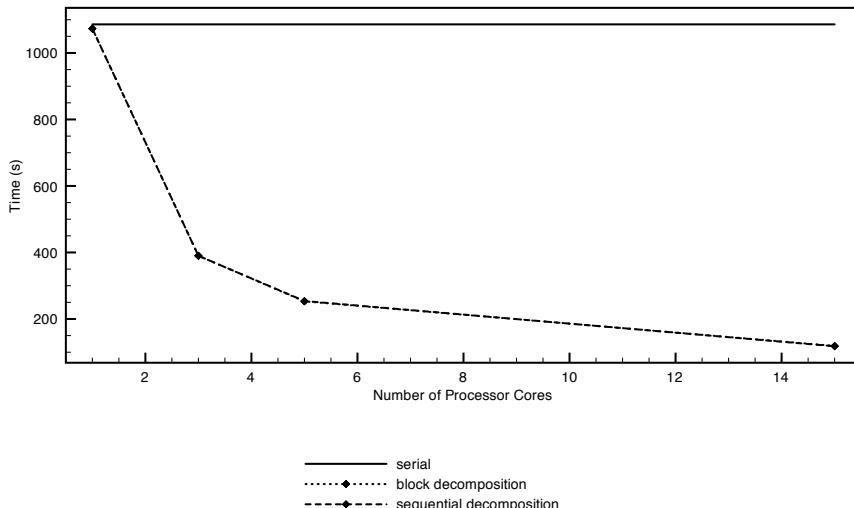


Fig. 4. CPU time vs. number of processor cores for CCl_4 (the block decomposition and sequential decomposition results are superimposed)

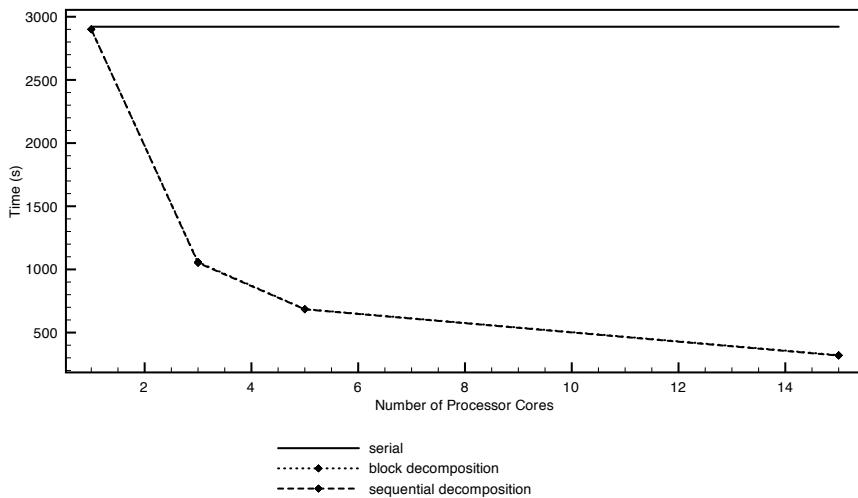


Fig. 5. CPU time vs. number of processor cores for CBr₄ (the block decomposition and sequential decomposition results are superimposed)

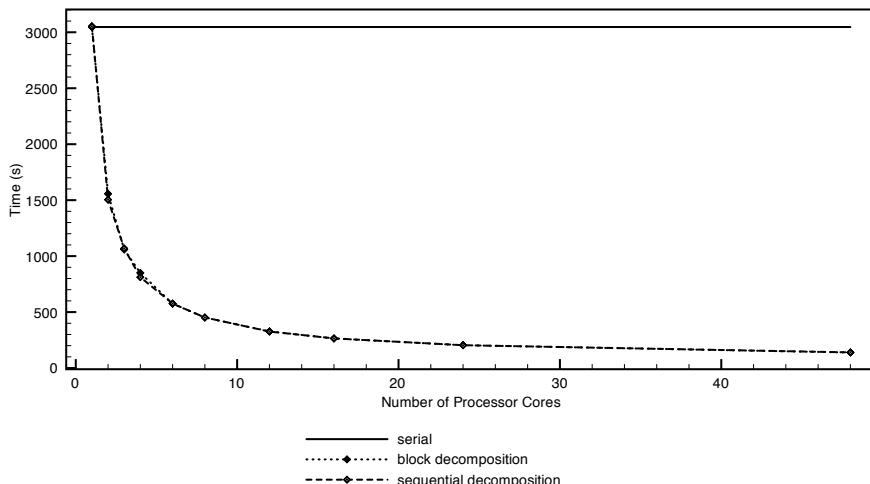


Fig. 6. CPU time vs. number of processor cores for C₅H₈ClFO (the block decomposition and sequential decomposition results are superimposed)

number of processor cores for C₅H₈ClFO (48), we get a speedup of approximately 22 times, from 3054 s to 140 s (see Table 3).

The average speedup that we are seeing with the three test structures (CCl₄, CBr₄, and C₅H₈ClFO) when the number of optimization parameters is the same as the number of processors used is 91%, with an average efficiency of 55%.

This is in contrast to the best performance improvement observed by Krishnan et. al. (6), which was a speedup of 90% with 4% efficiency. The difference in performance (efficiency and speedup) between our work and the work done by Krishnan et. al. can be attributed to a number of different factors. The first difference is the level of theory / method and the structure that was used. For our evaluation purposes, we used HF/6-31G(d) and three different structures, whereas Krishnan et. al. used MP2/cc-pVDZ and a single structure, $(\text{H}_2\text{O})_5$. The second difference relates to the efficiency of the code and the algorithm. Efficiency values published by Krishnan et. al. (6) show that their method has an efficiency of approximately 20% when running on 20 processor cores, whereas our algorithm has an efficiency of 62% when using 24 processor cores and 72% when using 16 processor cores for the structure $\text{C}_5\text{H}_8\text{ClFO}$. Further results relating to parallel fraction and efficiency can be seen in the next section (specifically Tables 1, 2, and 3).

5.2 Efficiency

Tables 1, 2, and 3 show the speedup, parallel fraction, and efficiency for each structure when the block and sequential decomposition methods are used. Our results show that 95–98% of the computational work is being completed in parallel with both the sequential and block based implementations on each of our test structures.

Table 1. Speedup, parallel fraction and efficiency for CCl_4 using sequential and block decomposition methods

CPUs	Sequential Decomposition						Block Decomposition					
	CPU Time (s)	S	f%	E%	CPU Time (s)	S	f%	E%				
1	1073	1.00	—	—	1073	1.00	—	—				
3	390	2.75	95	92	390	2.75	95	92				
5	254	4.22	95	84	254	4.22	95	84				
15	119	9.02	95	60	119	9.02	95	60				

Table 2. Speedup, parallel fraction and efficiency for CBr_4 using sequential and block decomposition methods

CPUs	Sequential Decomposition						Block Decomposition					
	CPU Time (s)	S	f%	E%	CPU Time (s)	S	f%	E%				
1	2901	1.00	—	—	2900	1.00	—	—				
3	1052	2.76	96	91	1058	2.74	95	91				
5	684	4.24	96	85	686	4.23	95	85				
15	318	9.12	95	60	321	9.03	95	61				

Table 3. Speedup, parallel fraction and efficiency for C₅H₈ClFO using block and sequential decomposition methods

CPUs	Sequential Decomposition				Block Decomposition			
	CPU Time (s)	S	f%	E%	CPU Time (s)	S	f%	E%
1	3054	1.00	—	—	3045	1.00	—	—
2	1554	1.97	98	98	1557	1.96	98	98
3	1069	2.86	97	95	1059	2.88	98	96
4	812	3.76	98	94	850	3.58	96	90
6	576	5.30	97	88	575	5.30	97	88
8	452	6.76	97	84	451	6.75	97	84
12	326	9.37	97	78	328	9.28	97	77
16	265	11.52	97	72	264	11.53	97	72
24	204	14.97	97	62	204	14.93	97	62
48	140	21.81	97	45	139	21.91	97	46

As can be seen in Tables 1, 2, and 3, the average parallel fraction remains relatively constant, whereas the efficiency decreases as more processors are used. This observed decrease in efficiency as more processors are used can be attributed to the additional inter-processor communication that is required.

From the investigation thus far, we see that our implementation scales reasonably well, and is parallelizing the calculations with a parallel fraction that is consistent with theoretical values, as discussed in Section 1.

5.3 Accuracy

As mentioned earlier, it is possible to introduce additional errors when performing calculations in parallel. To determine how accurate our calculations are, we compared four different frequency calculations, namely analytical, single processor numerical, and parallel numerical with both block and sequential decomposition.

First we compare the accuracy of the single processor numerical frequencies with the parallel processor numerical frequencies. This was done for all cases by calculating the largest and smallest percent difference between the frequencies (using both block and serial decomposition). For CCl₄ and CBr₄, the largest percent difference is 0.1% for all of the test calculations (results not shown). Therefore, the accuracy was not affected when moving from the single processor numerical implementation to the parallel numerical implementation. CBr₄ was chosen as the next step for Hessian calculations after CCl₄ as it increases the basis set size without increasing the number of atoms. Furthermore, CBr₄ is also important when considering accuracy as it includes a 3rd row element. For C₅H₈ClFO, the largest percent difference is 5.6%, and the average percent difference is 0.1% when comparing our parallel numerical implementation and the single processor numerical implementation. The results for the block and sequential methods are shown for different processor configurations in Table 4.

Table 4. Percent difference for single processor numerical frequencies and parallel numerical frequencies for C₅H₈ClFO. For percent differences $\geq 5\%$ the vibrational frequency is also included (cm^{-1}).

CPUs	Difference	Numerical (serial) vs Parallel Numerical	
		Block	Sequential
		Decomposition	Decomposition
48	Largest	1.5	1.5
48	Smallest	0.0	0.0
48	Average	0.1	0.1
24	Largest	0.3	2.8
24	Smallest	0.0	0.0
24	Average	0.0	0.1
16	Largest	0.7	2.8
16	Smallest	0.0	0.0
16	Average	0.0	0.1
12	Largest	0.5	4.0
12	Smallest	0.0	0.0
12	Average	0.0	0.1
8	Largest	0.0	5.6 (34.6)
8	Smallest	0.0	0.0
8	Average	0.0	0.2
6	Largest	1.5	2.3
6	Smallest	0.0	0.0
6	Average	0.1	0.1
4	Largest	0.0	1.0
4	Smallest	0.0	0.0
4	Average	0.0	0.1
3	Largest	1.0	3.0
3	Smallest	0.0	0.0
3	Average	0.0	0.1
2	Largest	0.0	5.3 (34.5)
2	Smallest	0.0	0.0
2	Average	0.0	0.2
1	Largest	0.0	0.0
1	Smallest	0.0	0.0
1	Average	0.0	0.0

Table 5. Percent difference for analytical frequencies compared with serial numerical, and parallel numerical frequencies for CBr₄

CPUs	Difference	Analytical vs		
		Serial Numerical	Parallel Block	Numerical Sequential
15	Largest Difference	1.7	1.7	1.7
	Smallest Difference	0.1	0.1	0.1
	Average Difference	0.8	0.8	0.8
5	Largest Difference	1.7	1.7	1.7
	Smallest Difference	0.1	0.1	0.1
	Average Difference	0.8	0.8	0.8
3	Largest Difference	1.7	1.7	1.7
	Smallest Difference	0.1	0.1	0.1
	Average Difference	0.8	0.8	0.8
1	Largest Difference	1.7	1.7	1.7
	Smallest Difference	0.1	0.1	0.1
	Average Difference	0.8	0.8	0.8

Next, we compared the frequencies produced by the three numerical methods (serial, block, and sequential) with the frequencies produced by an analytical method. Tables 5 and 6 show the largest, smallest, and average percent differences for all test cases involving CBr₄ and C₅H₈ClFO. When evaluating the results generated using CCl₄, the largest percent difference is 0.1%, and the average percent difference is 0.0% for all test cases (results not shown). For CBr₄ and C₅H₈ClFO, the largest percent difference is 1.7% and 9.0% respectively, and the average percent difference is 0.8% for CBr₄, and 0.7% for C₅H₈ClFO when compared with analytical frequencies.

The results obtained for the different frequency calculations show that there is minimal difference in the results obtained when comparing both the serial numerical and parallel numerical methods, as well as when comparing frequencies calculated numerically with those calculated analytically.

5.4 Hardware Comparison

The investigation thus far shows that our parallel method scales well, is accurate, and has a large parallel fraction. In order to further understand the performance and behaviour of our algorithm, thereby completing our investigation, we chose to extend our tests to include larger structures and different computer hardware configurations. The structures chosen for this component of our evaluation are C₅H₈ClFO (Figure 3), and C₉H₁₃N₃O₄ (Figure 7). As with the previous components of this evaluation, Cartesian coordinates and HF/6-31G(d) were used to compute all vibrational frequencies.

Table 6. Percent difference for analytical frequencies compared with serial numerical, and parallel numerical frequencies for C₅H₈ClFO. For percent differences $\geq 5\%$ the vibrational frequency is also included (cm^{-1}).

CPUs	Difference	Analytical vs		
		Serial	Parallel	Numerical
		Numerical	Block	Sequential
48	Largest	9.0 (32.7)	8.3 (33.0)	8.3 (33.0)
	Smallest	0.0	0.0	0.0
	Average	0.7	0.6	0.6
24	Largest	9.0 (32.7)	8.7 (32.8)	6.4 (33.7)
	Smallest	0.0	0.0	0.0
	Average	0.7	0.7	0.6
16	Largest	9.0 (32.7)	9.1 (32.7)	6.4 (33.7)
	Smallest	0.0	0.0	0.0
	Average	0.7	0.7	0.6
12	Largest	9.0 (32.7)	8.6 (32.9)	5.4 (34.0)
	Smallest	0.0	0.0	0.0
	Average	0.7	0.7	0.6
8	Largest	9.0 (32.7)	8.9 (32.8)	3.9
	Smallest	0.0	0.0	0.0
	Average	0.7	0.7	0.6
6	Largest	9.0 (32.7)	8.0 (33.1)	6.9 (33.5)
	Smallest	0.0	0.0	0.0
	Average	0.7	0.6	0.6
4	Largest	9.0 (32.7)	9.0 (32.7)	8.3 (33.0)
	Smallest	0.0	0.0	0.0
	Average	0.7	0.7	0.6
3	Largest	9.0 (32.7)	8.1 (33.1)	6.3 (33.7)
	Smallest	0.0	0.0	0.0
	Average	0.7	0.7	0.6
2	Largest	9.0 (32.7)	9.0 (32.7)	4.2
	Smallest	0.0	0.0	0.0
	Average	0.7	0.7	0.5
1	Largest	9.0 (32.7)	9.0 (32.7)	9.0 (32.7)
	Smallest	0.0	0.0	0.0
	Average	0.7	0.7	0.7

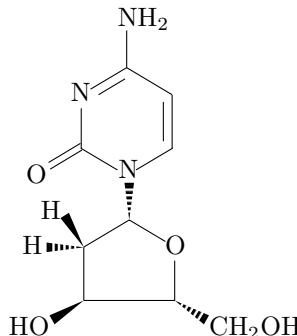


Fig. 7. Structure of C₉H₁₃N₃O₄

Our tests were conducted on systems manufactured by SUN Microsystems (with both Solaris and Linux operating systems), and Liquid Computing. The machines used for testing were configured so that there was adequate memory (RAM) to store all of the data associated with the test calculations. By storing the data in main memory we are able to better evaluate the performance of our parallel algorithm as calculations and application performance can be impacted by additional disk input and output operations.

Our hardware tests used three different compute clusters, namely, two clusters of SUN machines, and a machine designed by Liquid Computing. The two SUN clusters used for testing consisted of SUN Fire X4600 machines that were connected via Gigabit Ethernet. The only difference between the two is that one cluster uses SUN Solaris X86 as its operating system, and the other cluster uses Red Hat Enterprise Linux. In both cases our code was compiled using the SUN Studio compiler. On the Solaris cluster, SUN's HPC Cluster Tools was used to provide MPI communications, whereas on the Linux cluster, MPICH libraries that had been built with the Portland Group Compiler were used. It is important to note that each of the X4600 machines has 16 processor cores and at least 4GB of RAM per processor core. As a result, only calculations that use more than 16 processor cores utilize the Gigabit Ethernet connections for communications.

The machine designed by Liquid Computing was built using the LiquidIQ fabric computing architecture as a foundation. The LiquidIQ architecture employs a state-of-the-art hardware based backplane for communications and was configured the same way as the SUN Linux cluster mentioned above.

On the SUN Linux cluster and the Liquid Computing machine, we had access to sufficient processor cores to perform all calculations. On the SUN Solaris X86 cluster, the machines did not have enough memory available per processor core to enable us to determine the values for C₉H₁₃N₃O₄ when using 29 and 87 processor cores. As a result, these values have been approximated through the use of Amdahl's Law.

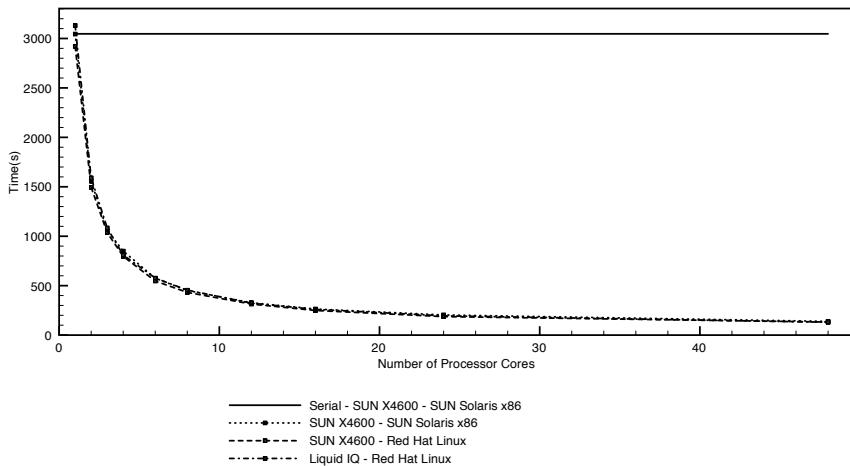


Fig. 8. CPU time vs. number of processor cores for C₅H₈ClFO, using block decomposition, on varying hardware / operating system configurations (results for the different configurations are nearly superimposed)

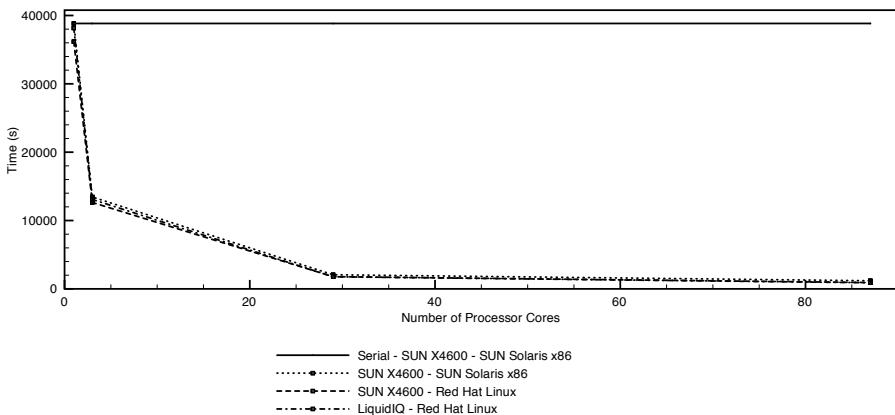


Fig. 9. CPU time vs. number of processor cores for C₉H₁₃N₃O₄, using block decomposition, on varying hardware / operating system configurations (results for the different configurations are nearly superimposed). Note that the results for the SUN X4600 - SUN Solaris x86 with 29 and 87 processors are approximated using Amdahl's law.

Using the two large test structures, we compared the CPU time of the calculation while varying the number of processors and the system configurations. Figures 8 (C₅H₈ClFO) and 9 (C₉H₁₃N₃O₄) illustrate the speedup and timing performance using the block version of our algorithm on the different clusters.

Table 7. Speedup, parallel fraction, and efficiency for C₅H₈ClFO using block decomposition on SUN X4600 and LiquidIQ hardware with different operating systems

CPUs	SUN Solaris x86				Red Hat Linux				Red Hat Linux			
	SUN X4600				SUN X4600				LiquidIQ			
	CPU		CPU		CPU		CPU		CPU		CPU	
	Time (s)	S	f%	E%	Time (s)	S	f%	E%	Time (s)	S	f%	E%
1	3045	1.00	—	—	2921	1.00	—	—	3131	1.00	—	—
2	1554	1.97	98.23	98	1495	1.95	97.64	98	1590	1.97	98.44	98
3	1069	2.86	97.50	95	1036	2.82	96.80	94	1084	2.89	98.07	96
4	812	3.76	97.88	94	797	3.66	96.95	92	810	3.87	98.84	97
6	576	5.30	97.37	88	547	5.34	97.53	89	577	5.43	97.89	90
8	452	6.76	97.37	84	432	6.76	97.38	85	455	6.88	97.68	86
12	326	9.37	97.45	78	314	9.30	97.36	78	323	9.69	97.84	81
16	265	11.52	97.41	72	249	11.73	97.57	73	260	12.04	97.81	75
24	204	14.97	97.38	62	187	15.62	97.67	65	195	16.06	97.85	67
48	140	21.81	97.45	45	132	22.13	97.51	46	130	24.08	97.89	50

Table 8. Speedup, parallel fraction, and efficiency for C₉H₁₃N₃O₄ using block decomposition on SUN X4600 and LiquidIQ hardware with different operating systems. Note that the results for the SUN X4600 - SUN Solaris x86 with 29 and 87 processors are approximated using Amdahl's law.

CPUs	SUN Solaris x86				Red Hat Linux				Red Hat Linux			
	SUN X4600				SUN X4600				LiquidIQ			
	CPU		CPU		CPU		CPU		CPU		CPU	
	Time (s)	S	f%	E%	Time (s)	S	f%	E%	Time (s)	S	f%	E%
1	38833	1.00	—	—	36173	1.00	—	—	38162	1.00	—	—
3	13447	2.89	98.06	96	12641	2.86	97.58	95	13081	2.92	98.58	97
29	2067	18.79	98.06	65	1785	20.26	98.46	70	1774	21.51	98.76	74
87	1192	32.58	98.06	37	926	39.06	98.57	45	897	42.54	98.78	49

These results show that there is minimal difference in timing between the block decomposition methods for all of our test structures. Figure 9 also shows that our numerical Hessian algorithm follows the same scaling trends as seen with the smaller test structures (Figures 4, 5 and 6).

Tables 7 and 8 contain the parallel fraction, speedup, and efficiency for C₅H₈ClFO and C₉H₁₃N₃O₄ as the number of processor cores being used was increased on the different hardware configurations.

The observed parallel fraction gives insight into communications and other performance differences that might be present between the different hardware / operating system configurations tested. Although the difference in parallel fraction between the different test machines is minimal (less than 1%), there is a larger difference observed when looking at speedup and efficiency.

For $C_9H_{13}N_3O_4$ we have a calculated speedup of 33 times and an efficiency of 37% when using 87 processor cores on the SUN Solaris machines. This is compared to a speedup of 39 times and an efficiency of 45% on the SUN Red Hat Linux machine, and a speedup of 43 times and an efficiency of 49% on the LiquidIQ machine for the same 87 processor test case. In the case of C_5H_8ClFO , a similar trend is observed. The calculated speedup for the SUN Solaris machine is 21 times with an efficiency of 45%. On the SUN Red Had Linux machine, a speedup of 22 times was observed, with an efficiency of 46%, and on the LiquidIQ machine a speedup of 24 times with an efficiency of 50% was observed. When reviewing the speedup and efficiency results, it can be seen that even the smallest difference in parallel fraction ($\leq 0.5\%$) can significantly impact efficiency.

For all structures, the frequencies calculated were numerically equivalent when comparing the different machines. In the case of 87 processor cores, the average percent difference is 0.3% when comparing the numerically generated frequencies to those that were generated analytically.

6 Summary

In this paper, we presented a software system independent algorithm that parallelizes and distributes the components of the numerical Hessian calculation. An implementation of our algorithm using MPI in conjunction with the MUNgauss computational chemistry software package was evaluated. This evaluation dealt with the following components: decomposition method, speedup, efficiency, accuracy, and a hardware / operating system comparison.

When comparing the single processor implementation with the sequential and block decomposition methods, results showed that there was minimal difference in terms of timing (performance), and parallel fraction (efficiency). Results also showed that we have a good ratio of parallel to serial work with a parallel fraction averaging 96%.

The accuracy of the frequencies calculated using the numerical Hessian method was very good, as the average percent difference between our parallel numerical method and a serial numerical method is 0.1%. This high level of accuracy is also maintained when comparing our parallel numerical frequencies with frequencies that have been computed analytically, where the average percent difference is 1%. Our observed percent differences, obtained through central difference calculations, show that our numerical results have a high enough accuracy to be used when an analytical method is not available.

The investigation was continued with a hardware / operating system comparison. Results showed that our algorithm performs consistently on different hardware platforms and on different operating systems. Although, users wanting to optimize performance, can make certain choices in hardware components and operating systems that will affect the efficiency by as much as $\pm 5\%$. The best performance results obtained for our largest structure ($C_9H_{13}N_3O_4$) using 87 processor cores was obtained using the LiquidIQ machine with Red Hat Linux and Sun Studio compiler. In this case our implementation of the parallel algorithm outperformed the equivalent single-processor version by a factor of 98%.

The results presented in this paper show that a performance benefit can be obtained, without impacting numerical accuracy, through the distribution of the components of the numerical Hessian calculation across several processors. Furthermore these results assist with the establishment of minimum performance and accuracy expectations when using commercial software that performs parallel numerical Hessian calculations.

Acknowledgements

This research would not have been possible without support from the Atlantic Computational Excellence Network (ACEnet), the Natural Sciences and Engineering Research Council of Canada (NSERC), and Liquid Computing.

Computational facilities were provided by ACEnet and Liquid Computing Inc.

ACEnet is the regional high performance computing consortium for universities in Atlantic Canada, and is funded by the Canada Foundation for Innovation (CFI), the Atlantic Canada Opportunities Agency (ACOA), and the provinces of Newfoundland & Labrador, Nova Scotia, and New Brunswick.

References

- [1] Pople, J.A., Krishnan, R., Schlegel, H.B., Binkley, J.S.: Derivative Studies in Hartree-Fock and Møller-Plesset Theories. *Int. J. Quant. Chem.* 16(S13), 225–241 (1979)
- [2] Schlegel, H.: Estimating the Hessian for Gradient-Type Geometry Optimizations. *Theoretica. Chimica. Acta* 66(5), 333–340 (1984)
- [3] Poirier, R.A., Hollett, J.: MUNgauss (Fortran 90 version). Chemistry Department, Memorial University of Newfoundland, St. John's, NL, A1B 3X7 (2007); With contributions from Bungay, S.D., El-Sherbiny, A., Gosse, T., Keefe, D., Kelly, A., Pye, C.C., Reid, D., Saputantri, K., Shaw, M., Staveley, M.S., Wang, Y., Xidos, J.
- [4] Amdahl, G.: Validity of the single processor approach to achieving large-scale computing capabilities. In: Proceedings of the American Federation of Information Processing Societies, Washington, DC, vol. 30, pp. 483–485. AFIPS Press (1967)
- [5] Frisch, M.J., Trucks, G.W., Schlegel, H.B., Scuseria, G.E., Robb, M.A., Cheeseman, J.R., Montgomery Jr., J.A., Vreven, T., Kudin, K.N., Burant, J.C., Millam, J.M., Iyengar, S.S., Tomasi, J., Barone, V., Mennucci, B., Cossi, M., Scalmani, G., Rega, N., Petersson, G.A., Nakatsuji, H., Hada, M., Ehara, M., Toyota, K., Fukuda, R., Hasegawa, J., Ishida, M., Nakajima, T., Honda, Y., Kitao, O., Nakai, H., Klene, M., Li, X., Knox, J.E., Hratchian, H.P., Cross, J.B., Adamo, C., Jaramillo, J., Gomperts, R., Stratmann, R.E., Yazyev, O., Austin, A.J., Cammi, R., Pomelli, C., Ochterski, J.W., Ayala, P.Y., Morokuma, K., Voth, G.A., Salvador, P., Dannenberg, J.J., Zakrzewski, V.G., Dapprich, S., Daniels, A.D., Strain, M.C., Farkas, O., Malick, D.K., Rabuck, A.D., Raghavachari, K., Foresman, J.B., Ortiz, J.V., Cui, Q., Baboul, A.G., Clifford, S., Cioslowski, J., Stefanov, B.B., Liu, G., Liashenko, A., Piskorz, P., Komaromi, I., Martin, R.L., Fox, D.J., Keith, T., Al-Laham, M.A., Peng, C.Y., Nanayakkara, A., Challacombe, M., Gill, P.M.W., Johnson, B., Chen, W., Wong, M.W., Gonzalez, C., Pople, J.A.: Gaussian 2003, Revision B.05. Gaussian, Inc., Pittsburgh, PA (2003)

- [6] Krishnan, M., Alexeev, Y., Windus, T.L., Nieplocha, J.: Multilevel parallelism in computational chemistry using common component computer architecture and global arrays. In: Conference on High Performance Networking and Computing. Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, Seattle, WA (2005)
- [7] Bylaska, E.J., de Jong, W.A., Kowalski, K., Straatsma, T.P., Valiev, M., Wang, D., Apra, E., Windus, T.L., Hirata, S., Hackler, M.T., Zhao, Y., Fan, P.D., Harrison, R.J., Dupuis, M., Smith, D.M.A., Nieplocha, J., Tippuraju, V., Krishnan, M., Auer, A.A., Nooijen, M., Brown, E., Cisneros, G., Fann, G.I., Fruchtl, H., Garza, J., Hirao, K., Kendall, R., Nichols, J.A., Tsemekhman, K., Wolinski, K., Anchell, J., Bernholdt, D., Borowski, P., Clark, T., Clerc, D., Dachsel, H., Deegan, M., Dyall, K., Elwood, D., Glendening, E., Gutowski, M., Hess, A., Jaffe, J., Johnson, B., Ju, J., Kobayashiand, R., Kutteh, R., Lin, Z., Littlefield, R., Long, X., Meng, B., Nakajima, T., Niu, S., Pollack, L., Rosing, M., Sandroneand, G., Stave, M., Taylor, H., Thomas, G., van Lenthe, J., Wong, A., Zhang, Z.: NWChem, A computational chemistry package for parallel computers, Version 5.0. Pacific Northwest National Laboratory, Richland, Washington 99352-0999, USA (2006)
- [8] Kendall, R., Apra, E., Bernholdt, D.E., Bylaska, E.J., Dupuis, M., Fann, G.I., Harrison, R.J., Ju, J., Nichols, J.A., Nieplocha, J., Straatsma, T.P., Windus, T., Wong, A.T.: High performance computational chemistry: An overview of NWChem a distributed parallel application. Computer Phys. Comm. 128, 260–283 (2000)

Preliminary Findings of Visualization of the Interruptible Moment

Edward R. Sykes

School of Applied Computing and Engineering Sciences
Sheridan College Institute of Technology and Advanced Learning
1430 Trafalgar Road, Oakville, Ontario, Canada
ed.sykes@sheridanc.on.ca

Abstract. Intelligent software must frequently interrupt users, but the problem of deciding when to interrupt the user is still unsolved. In this paper, an algorithm is presented that identifies the appropriate times to interrupt the user is proposed and visualizations of the user's state. The Intelligent Interruption Algorithm draws from user, task, and environmental contextual information dynamically extracted from the environment as the user performs computer based tasks. The visualizations are a representation of these complex components presented in a way that is meaningful for analysis purposes and for the user. This paper presents the following: the interruption algorithm; the machine learning algorithms used; and the preliminary results of visualizing the interruptible moment.

Keywords: Visualization, HCI, user models, HPC and real-time user modeling.

1 Introduction

Algorithmically deciding when to interrupt a user as s/he performs computer-based tasks is an unsolved problem. A number of researchers are currently working on this problem but it is very difficult because there are a number of dependent problems that are also extremely difficult to solve [1-3]. Examples of these related problems include accurately ascertaining the user's goal as s/he is performing the task; assessing how difficult or easy the task is for the user; and estimating the cost of the interruption and resumption lag [4-10].

I propose the design of an algorithm that reasons about interruption scenarios and makes accurate decisions about when to interrupt the user a solution to these problems. [11-13]. The algorithm I propose uses two different reasoning approaches: a Dynamic Bayesian Network and an Adaptive Neural-Based Fuzzy Inference System, both novel approaches to predict interruption times. This research extends Gievská and Sibert's research by adding to the interruption taxonomy and validates the Bayesian Network used for interruption timings [1]. Most research to date has focused exclusively on the task dimension [1, 3, 14]. The algorithm in this research draws information from all three dimensions (i.e., user, task, and environment) to reason about deciding optimal times to interrupt the user. Another more general benefit of this research is that an interruption algorithm could be applied to numerous domains

and applications. Personal digital assistants, smartphones, office software suites, workflow planning software, and mixed-initiative interaction systems could benefit from an accurate holistic interruption algorithm.

The use of High Performance Computing is applied in this research in several ways. The tools Paraview and MATLAB were used for visualizing the current user state using real-time contextual data while relying on the computational resources of SHARCNET clusters¹. Furthermore, parallel implementations were explored for network learning for the Bayesian implementation of the interruption algorithm [15].

The Intelligent Interruption Algorithm draws from a user model consisting of user, task and environments. This paper is a discussion of the interruption algorithm; the machine learning algorithms used; and the preliminary results of visualizing the interruptible moment.

2 Intelligent Interruption Algorithm

This section discusses the Intelligent Interruption Algorithm in terms of the desired characteristics from a design perspective; the machine learning algorithms selected with rationale for choosing them; the parameters representing the types of data collected; and how high performance computing was used to assist in solving this problem.

2.1 Intelligent Interruption Algorithm Desired Characteristics

The Intelligent Interruption Algorithm draws from user profile information and real-time observations of the user's actions as s/he performs a task in the environment. In designing the algorithm, I focussed on certain desirable characteristics such as making accurate decisions when uncertainty is present and being computationally efficient in order to make interruption decisions in real time. In other words, the algorithm must be able to decide whether or not to interrupt a user within a two second window of time—if an algorithm is not capable of making decisions within this timeframe then it would not be satisfactory. I employ a user model (e.g., preferences, familiar tasks, etc.) that will be used in the interruption decision-making process and it needs to draw on direct measurements from user activities.

From the human side of the design, the algorithm must be able to respond to non-linear inputs (i.e., emotions). This characteristic is based on psychological research that has shown that human emotions can escalate very quickly and demonstrate non-linear growth rates (e.g., frustration, anxiety, etc.) [16] The algorithm's decision-making process to interrupt or defer an interruption must be easy to be examined and understood by a human. This characteristic provides the opportunity for deeper reasoning into why an interruption occurred as well as insight into the form and content of an appropriate interruption message. The algorithm must be capable of accepting input-output patterns and learning these associations. This is because the algorithm is

¹ Paraview is an open-source, multi-platform visualization tool that can be used to analyze extremely large datasets using supercomputers (source: <http://www.paraview.org/>). MATLAB's Parallel Computing Toolbox solve computationally and data-intensive problems on multi-core and multiprocessor supercomputers (source: <http://www.mathworks.com/products/parallel-computing/>)

modeling it's decision making process after a human tutor; and it learns quickly from a small number of training data sets. Pragmatically, there will be a limited number of participants in this research; as a result, there will be a relatively small number of user-interruption session data available from which the algorithm must learn.

Collectively, these desired characteristics shaped the design of the Intelligent Interruption Algorithm and served to frame the appropriateness of the types of machine learning algorithms that were employed and how the interruption data were effectively used. The following section discusses the parameters (variables) representing the types of data the algorithm requires in relation to an interruption taxonomy developed by Gievska and Sibert [1].

2.2 Algorithm Parameters and the Interruption Taxonomy

I used Gievska and Sibert's interruption taxonomy as a conceptual framework to identify attributes and relationships appropriate for conceptualizing factors that influence the timing of interruptions. Figure 1 represents the three dimensions: User, Environment, and Task contexts [1]. Parameter data from all three dimensions of the interruption taxonomy are important and need to be represented in the interruption decision-making process.

My model extends the taxonomy by adding two more attributes to incorporate personalization into a learning task. Personality Type (Learning Styles) is crucial as there is a significant body of literature that supports each of us have a different (and preferred) learning style [17]. Another enhancement to the taxonomy is the inclusion of the user's current cognitive load level as s/he is working on a task. These two enhancements need to be incorporated into the decision making of an interruption algorithm.

The above-mentioned high-level parameters are collected and used by the Intelligent Interruption Algorithm (presented in bold in Figure 1).

The first dimension is **User Context**. The *user profile* represents the user's capability level of working on computers, his/her ability to concentrate on tasks for long periods of time, frustration level, distraction level, familiarity with the tasks, etc. This information will be captured before the experiment by using a comprehensive participant questionnaire that later is fed into the algorithm. The profile also includes the *UsersTaskFocus*, a composite parameter representing how focused the user is on the specifics of the current task. Included in the measurement are the user's eye measurements and movements via FaceLAB (eye tracking hardware and software); facial features and changes; head movements; and computer interactions.

For the experiment constructed in this research (discussed in section 3), participants were asked to perform the computer-based task as quickly and as accurately as possible. Using the measurements on eye movements and more specifically, the exact locations of where on the screen the user is concentrating his/her efforts and for how long afforded much information as to how focused the user was with reference to completing the task [18]. Data from FacialFeatures, HeadMovements and PhysicalIdleTime (how much time has elapsed since the user's last computer interaction) were used in the decision making process too. Data from these inputs were used to determine if the user was frowning or confused, possibly indicating frustration with the task. The last set of data captured (i.e., Mouse_movement, mouse_clicks, and User_Typing) were used to measure task progress and relative position the user was

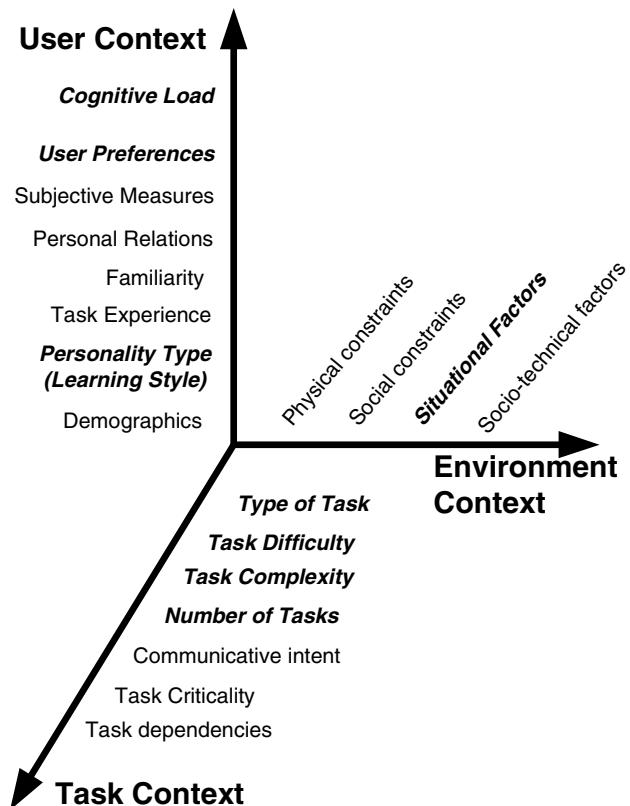


Fig. 1. Interruption Taxonomy [1]

in the task (i.e., early, mid, or late). This information was used to compare the user's current progress against a normative model of a typical user to complete the task and measures the user's on-task progress.

The second dimension of our Interruption Taxonomy is **Task Context**, which is comprised of Task Difficulty, SubjectiveComplexity (i.e., Task Familiarity, Preferred Task), the CurrentSubtask and the PreviousSubtask. The algorithm to monitor how the user is progressing through the task uses the measurements in the Task Context with reference to the particulars of the task. For example, if the algorithm believes that important information should be disclosed to the user and just detected that a user subtask boundary, it would choose this point as an interruption point [1, 3].

The third dimension is the **Environment Context** which deals with information that represents the physical and dynamic environment where human-computer interaction occurs [1]. By including environment-related attributes significantly higher sensitivity can be achieved in the system so that it can respond appropriately to changes in social, physical settings and the current situation [1]. In this research, the significant parameter in this dimension is the **SituationalFactors**. It is a top-level parameter dependent on the following data: "expected time to complete," "expected performance outcome," and "temporal properties" [1]. Collectively, these parameters

capture salient factors influencing time constraints (implicit, and explicit), risk, and cognitive load.

2.3 Extraction and Representation of the Contextual Data

The Intelligent Interruption System components include Task and Environment Models, a User Model, and the Intelligent Interruption Algorithm that draws contextual information from each of these three models. A Data Management Module houses two core units: the Task and Environment Model and the User Model. The Task and Environment model represents the details regarding the task and situational factors (environment) for the user. The User Model is divided into two units: a dynamic user model (for real-time data) and a static user model (for unchanging or slowly changing data). All of the data in the Data Management Module are represented in the concatenation of the three vectors, \vec{v} , \vec{w} and \vec{x} . The Dynamic User Model contains vector \vec{v} which represents live (real-time) data consisting of the user's current state and activities (e.g., determining if the user is reading, or clicking the mouse, etc.). The Static User Model holds vector \vec{w} representing the static user data from external data sources (e.g., user profile, learning style, etc.). The Task Model represented by vector \vec{x} contains task- and environmental-specific information (e.g., expected time to complete, subtask complexity, etc.). This vector is populated in advance of the user being introduced to the task. The Intelligent Interruption Algorithm module uses the Data Management Module by extracting the vector values for its use. The collection of data (i.e., \vec{v} , \vec{w} and \vec{x}) is captured and fed into the Intelligent Interruption Algorithm. If the Intelligent Interruption Algorithm decides this is an optimal time to interrupt the user, then the user is interrupted.

2.4 Dynamic Bayesian Network Architecture

Dynamic Bayesian Networks offer numerous benefits to an algorithm designed to solve the proposed research problem as they have the ability to handle incomplete data as well as uncertainty; they are trainable and provide means for avoiding overfitting; they encode causality in an intuitive graphical fashion; they offer a framework for combining prior knowledge and data; and they are modular and parallelizable [15, 19].

In this research, I am primarily interested in Bayesian Networks to have the optimal decision at the top level of the network (i.e., to interrupt or not interrupt the user). Components of the network are based on Gievska and Sibert's interruption research—specifically the TaskDifficulty subtree [1]. Figure 2 shows the initial design of the Dynamic Bayesian Network.

In the bottom right section of Figure 2, two sub trees represent the *StrengthofRelationship* between two subtasks. This subtree is used to assist in the detection of task boundaries with the intent that interruptions at task boundaries may be appropriate if these tasks are different [20]. Iqbal and Bailey's research showed that the Cost of Interruption is much reduced if the interruption match task boundaries [21].

In order to represent the user's focus on the task at hand, the parameters *FrustrationLevel* and *Distractibility* are used in modeling this part of the interruption decision network. The top left nodes of Figure 2 represent the *UsersTaskFocus* sub tree (the majority of data for this sub tree will come from the Eye Tracking system).

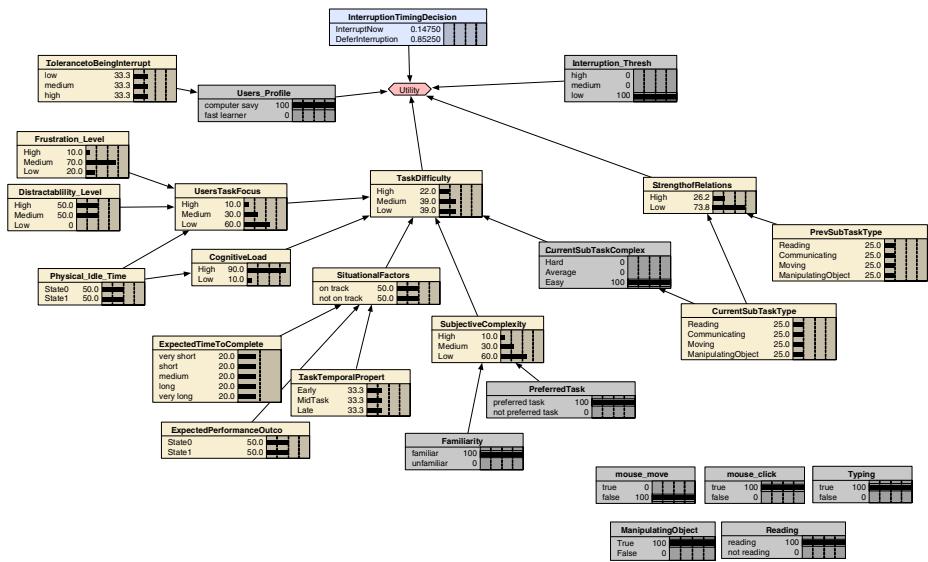


Fig. 2. Initial Design of the Dynamic Bayesian Network

2.5 Adaptive Neural-Based Fuzzy Inference System Architecture

I have also incorporated the initial design of the Fuzzy Inference System in the Intelligent Interruption Algorithm. This was developed with the input of several researchers at the Sheridan Institute of Technology and Advanced Learning. Synthetic sample training data was constructed and used with the Fuzzy Inference System to create the Adaptive Neural-Based Fuzzy Inference System. This Adaptive Neural-Based Fuzzy Inference System model is shown in Figure 3 and the rules for the system are shown in Figure 4.

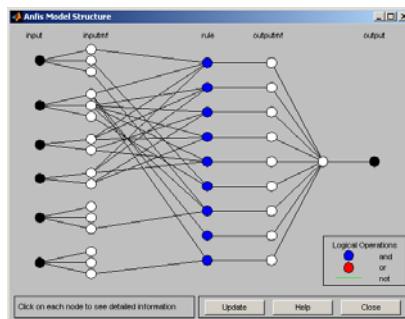


Fig. 3. Initial Design of the Adaptive Neural-Based Fuzzy Inference System

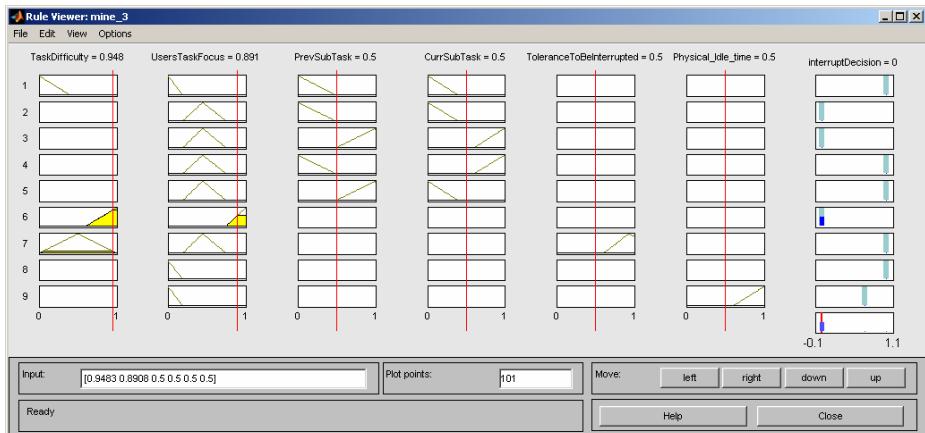


Fig. 4. Rule Viewer for the initial design of the Adaptive Neural-Based Fuzzy Inference System showing that if the *TaskDifficulty* is **high** and the *UsersTaskFocus* is currently **high**, then *InterruptionDecision* is set to **no** (i.e., defer the interruption).

Like Dynamic Bayesian Networks, Adaptive Neural-Based Fuzzy Inference Systems offer numerous benefits. In relation to designing a solution to the proposed research problem, Adaptive Neural-Fuzzy Inference Systems provide an intuitive way in which a domain expert's knowledge can be embodied within a Fuzzy Inference System. This is done by extracting and describing the knowledge using linguistic variables and membership functions [22-24]. Using the Fuzzy Inference System as an initial design for modeling the problem domain, the Adaptive Neural-Based Fuzzy Inference System can then use it to tune and refine the model during the training phase. Adaptive Neural-Fuzzy Inference Systems offer other benefits too. They require a smaller training set to converge when compared to other machine learning tools and have gained a wide level of acceptance from academics to industry specialists [22-24]. Furthermore, they are easily inspectable and interpretable by humans. This last point offers a natural extension to this research that is to provide a meaningful message to the user once an interruption point has been established. Because the Adaptive Neural-Fuzzy Inference System's reasoning process is easily interpretable by a human, insight into the construction and delivery of a message becomes a very feasible expansion of this research.

3 Experiment Design

The goal of the Intelligent Interruption Algorithm is to weigh the available information presented and determine if it should interrupt the user or defer interruption until a more opportune time. The experiment for this research is based on the following context. Given a user working on a computer based task comprised of subtasks, a human tutor will observe the user's progress and interrupt him/her when s/he feels such an interruption would be beneficial. During these sessions, the user may also initiate interruptions at any time. This experiment is repeated for 3 to 6 users with the same

tutor to collect data for all user-tutoring sessions. Each tutoring session data includes all the contextually relevant information as the user performs the assigned computer based task. These data, with the tutor's interruption timings, will then be fed back into the two Intelligent Interruption Algorithm implementations (Dynamic Bayesian Networks, and Adaptive Neural-Fuzzy Inference System) for comparison purposes to determine the degree of accuracy each algorithm offers for each interruption type (user-initiated, or tutor-initiated). The details of the experiment method are described in Table 1.

Table 1. Steps involved in the experiment procedure

Step	Activity	Approximate Time
1	Participant Opening Questionnaire: A questionnaire obtaining information on participant background, frustration level, demographics, etc.	5 minutes
2	Tutor Opening Questionnaire: A questionnaire obtaining information on domain expert demographics and background in relation to interruption expertise (e.g., tutoring, teaching, etc.)	5 minutes
3	Participant Briefing: The participant is briefed about the experiment and the computer-based tasks. For calibration purposes, the participant will perform an orientation session. The participant will also be briefed about their rights as per the approved ethical protocol.	5 minutes
4	Task activity: The participant performs the task using the designated computer and equipment (e.g., eye tracker). The human tutor may interrupt participants when s/he feels that such an interruption would be beneficial. The participant may also interrupt the human tutor to request clarification or more information. The interruption timings will be recorded with all related contextually relevant parameters in a log file.	10 minutes
5	Participant Closing Questionnaire: This questionnaire is used to collect each of the participant's comments regarding the tasks, the interruptions that occurred, the tutor's behaviour and advice, etc. Brief, informal interviews will also be conducted where necessary to obtain clarification on questionnaire responses.	5 minutes (per participant)
6	Tutor Closing Questionnaire: This questionnaire is used to collect each of the tutor's comments regarding the tasks, the interruptions that occurred, the tutor's behaviour, advice, etc. Brief, informal interviews will also be conducted where necessary to obtain clarification on questionnaire responses.	5 minutes (per domain expert)
7	Repeat steps 1-6 for all the participants and domain experts in the experiment (between three and six participants each for the training data and test data)	
8	Wrap-up: Thank the participants and tutors for their time and contribution.	1 minute

3.1 The Experimental Task for Participants

In designing appropriate tasks for this research, consideration was given to a number of factors. For example, a good task is one in which the learning curve is minimized

for the user and the task is not unnecessarily complicated for the user to perform; the significance of the interruption and the timing or appropriateness of the task (i.e., 15 minutes or less) to increase the probability that the user would remain focused (both mentally and ocularly) on the task. I also took into account the suitability for measurement and the suitability for use with the eye tracking system. This is a technical requirement based on the limitations of the system resources that the eye tracking system is currently using. The eye tracking system is very demanding on the computer, thus the task selected cannot overburden an already laboured system.

These requirements were carefully weighed in the selection process of an appropriate task. A number of tasks satisfy these requirements, but the game of Sudoku has a number of beneficial properties that made it a good choice for the task used in this research. Sudoku is a popular and easy to understand puzzle. The game can be designed to be quite simple or quite difficult. From an eye tracking and eye movement perspective most of the effort is relative to the 3x3 square the user is currently working on, however, a degree of eye movement is required to scan the entire board in order to complete the current square. Sudoku provides a controlled environment to divide up the area into subtasks that can be used to determine whether or not the user is attending to these subtasks based on his/her real-time attention focus.

In reference to the experiment, Figure 5 shows the computer-based task for this experiment. There were 7 trials (games) of the same level of difficulty. The participant was asked to complete regions S_1 and S_2 as quickly and as accurately as possible. The task difficulties of S_1 and S_2 are comparable. The human tutor had the initial Sudoku game sheet and the answers so as to offer help to the participant if needed. The games selected had only one solution. Timings were based on the eye-tracking software, which offers 5 milliseconds accuracy.

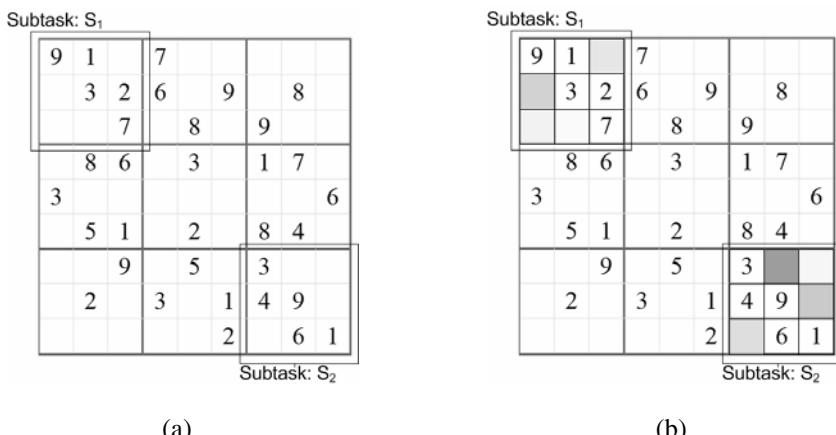


Fig. 5. (a) Sudoku—The computer-based task used in the experiment, and (b) with LookZones representing increased granularity of the user's attentional focus within the two SubTasks (S_1 and S_2).

4 Visualization Analysis Tools

The motivation for visualizing data in this problem is to more clearly understand the relationship between the various parameters that influence the interruption point. A sample set of user-system interaction data was used in this study. The purpose of this testing was to ascertain the accuracy of the Dynamic Bayesian Network and Adaptive Neural-Based Fuzzy Inference System implementations and to properly tune them appropriately.

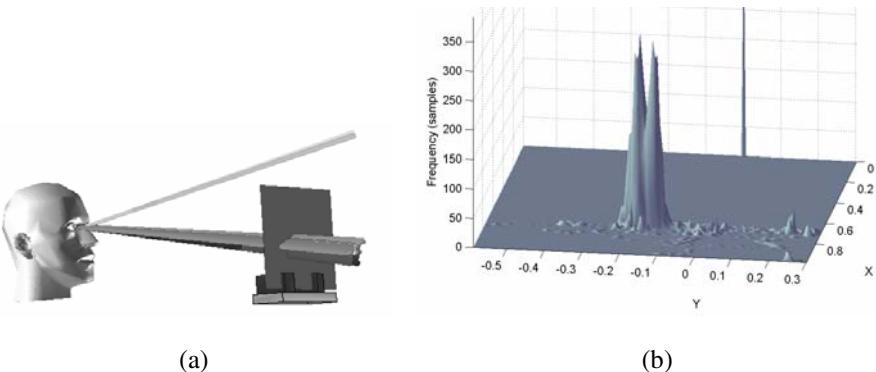


Fig. 6. Eye Tracking Data Brower Analysis tool: (a) 3D replay (left and right eyes, and head tilt) and (b) 2D histogram of two parameters (left eye and right eye pupil diameters)

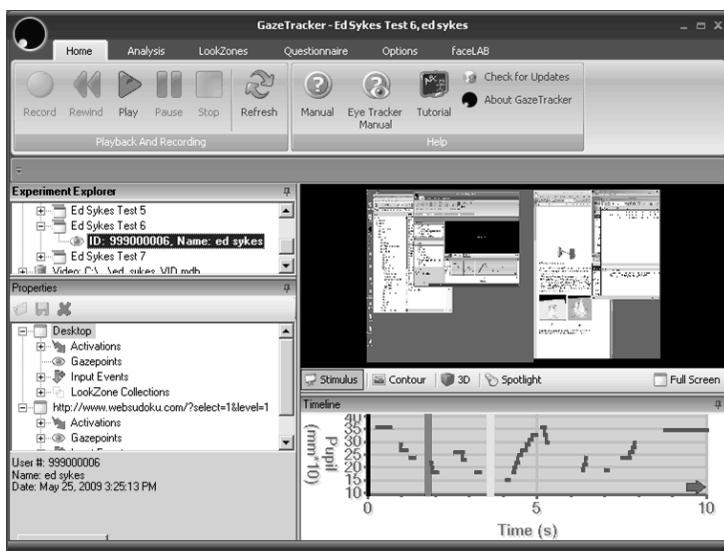


Fig. 7. GazeTracker Analysis tool. The bottom right screen depicts a graph of events (mouse clicks, keys pressed, etc.) that occurred over time against pupil diameter.

The data were analyzed using a number of different software packages. The Eye Tracking Data Browser (shown in Figure 6) provided various ways to visualize the data for: eye movements; head positions and rotations; eyelid aperture; lip and eye-brow movements, and pupil diameter [18]. Another software package used was the GazeTracker Analysis tool (shown in Figure 7). This tool was used to analyze and visualize the different types of data that GazeTracker collects (e.g., video and image stimuli analysis). GazeTracker provides powerful tools for navigating and interpreting all of its captured data (e.g., saccades², perclos, fixations, gazepoints, percentage of time the user focused on SubTasks, etc.).

5 Discussion

This section discusses the findings from visualization analysis of the user-tutor interruption data that was used by the Intelligent Interruption Algorithm and the conclusions.

5.1 Findings from Visualization Analysis

Contour maps were constructed for several of the datasets including mesh graphs using MATLAB. Figure 8 shows the results of the attentional focus of the user as they proceeded to solve the first Sudoku problem (please refer to Figure 5 (a)). In Figure 8 (a), the user has completed Subtask S₁ only. The visual and data analysis for this user's performance on this Subtask shows that the user remained focused on the task (84% of the total attentional eye focus was within the Look Zone boundaries that demark S₁).

In Figure 8 (b), the user has finished Subtask S₁ and Subtask S₂. The visual and data analysis reveals that the user remained focused for both Subtasks (78% of the total attentional eye focus was within S₁ and S₂ Look Zone boundaries). However, additional time and effort was needed for S₂ (16% beyond Subtask S₁). If this trend were to continue, the Intelligent Interruption Algorithm would consider interrupting the user to offer advice.

In another trial involving Sudoku, the participant was asked to complete both S₁ and S₂ as quickly and accurately as possible, with interesting results. The contour map shown in Figure 9 illustrates the user's attentional focus is unbalanced between S₁ and S₂ and the rest of the Sudoku board. This visualization shows that the participant spent significantly more effort and time on Subtask S₂ than on S₁. In fact, the user spent 79% of their attentional focus in S₂ but a considerable amount of effort was placed in the bottom left cell of S₂. The visualization also shows that the user's attentional focus at times is not in relation to either Subtask. In fact, 22% of the total attentional focus was spent outside of S₁ and S₂. These observations in combination with

² A saccade is an abrupt quick but short movement in both eyes. The Eye Tracker records major saccades (noticeable with the naked eye) and minor saccades (where special instrumentation is needed to detect them).

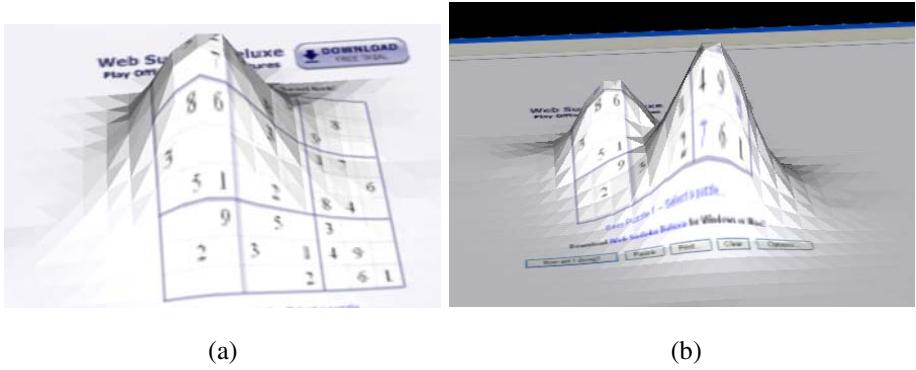


Fig. 8. (a) Modeling the user, task, and environment context. A contour map of the attentional focus of a user working on completing the top-left square of a Sudoku puzzle (SubTask S₁); (b) A contour map of the user's attentional focus after completing both SubTask S₁ and S₂. The user remained focused for both Subtasks, however, additional time and effort was placed on SubTask S₂.

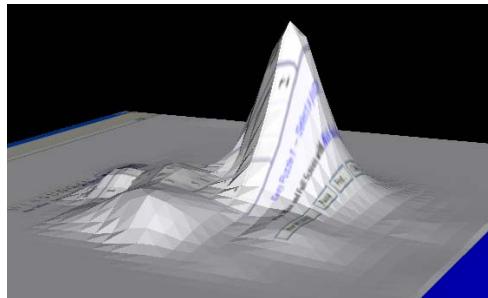


Fig. 9. A potentially good interruption point: The participant has spent a significantly more amount of effort and time on Subtask S₂ than on S₁; much attentional focus was spent on the bottom left cell in S₂; and a significant amount of attentional focus was not spent on either SubTask S₁ or S₂.

various other parameter inputs (as discussed in Section 2), would make the Intelligent Interruption Algorithm reasoning system to strongly consider this point in time as a good interruption point.

Other visualizations were also created. For example, meshes were constructed using MATLAB and SHARCNET to provide additional insight into the user's actions and intentions in relation to deciding interruption points. Figure 10 shows one mesh where the most prevent change in the user's eye behaviour was blink rate and the timings of saccades. This insight provided significant guidance in terms of tuning the Intelligent Interruption Algorithm to be sensitive to these specific input parameters.

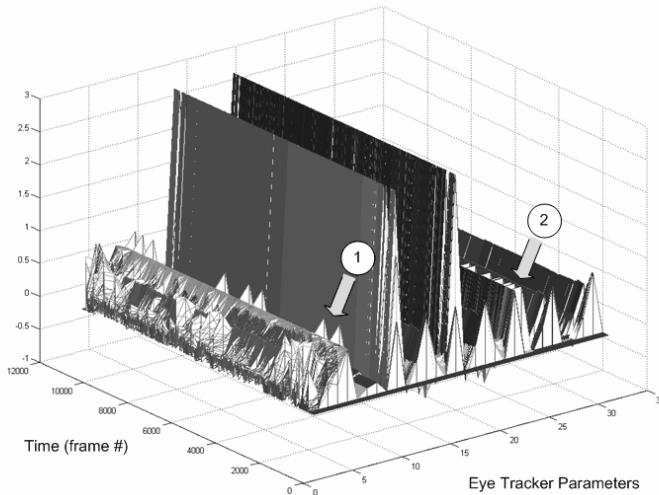


Fig. 10. Mesh rendering of the Eye Tracking Parameters using MATLAB and SHARCNET. (1) represents blink states for the user, and (2) saccades.

5.2 Conclusions

This paper presented the motivation for an Intelligent Interruption Algorithm, machine learning algorithms implemented and the preliminary findings of visualizing the user's state as it relates to assisting in the determination of good interruption points.

The Intelligent Interruption Algorithm draws from user, task, and environment contextual information dynamically extracted from the environment as the user performs computer-based tasks. The visualizations of the collected data were insightful in identifying interruption points that were not obvious through traditional statistical analysis methods. Through visualization analysis, it became evident when the algorithm should interrupt a user by identifying specific parameters and data from the three dimensions in the taxonomy. For instance, placing particular emphasis on the user's attentional focus in relation to the task proved to be extremely important. When the user's focus deviated by an automatically learned threshold, the Intelligent Interruption Algorithm triggered an interruption point. Other parameters, such as *perclos*, pupil diameter, blink rate, saccades, and mouse and keyboard events also underwent visual analysis but it was found that these parameters were not as significant as the user's attentional focus in determining good interruption points. Collectively, these visualizations provided the means by which careful refinement to the Intelligent Interruption Algorithm occurred.

The Intelligent Interruption Algorithm is unique in that it could easily be incorporated into an agent for the purpose of personalized tutoring, or to augment electronic devices such as smartphones, or to improve mixed-initiative interaction systems. While these findings are preliminary in nature, I am confident that they are a contribution to the field. Additional experiments and developments to the Intelligent Interruption Algorithm are in progress.

Acknowledgements. This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca).

References

1. Gievská, S., Sibert, J.: Using Task Context Variables for Selecting the Best Timing for Interrupting Users. In: Smart Objects and Ambient Intelligence Conference, Grenoble, France, pp. 171–176 (2005)
2. Horvitz, E., Apacible, J.: Learning and Reasoning about Interruption. In: ICMI. CHI, pp. 20–27 (2003)
3. Iqbal, S., Bailey, B.: Understanding and Developing Models for Detecting and Differentiating Breakpoints during Interactive Tasks. In: CHI 2007, pp. 697–706. ACM, San Jose (2007)
4. Conati, C., Zhao, X.: Building and Evaluating an Intelligent Pedagogical Agent to Improve the Effectiveness of an Educational Game. In: IUI. ACM, Madeira (2004)
5. Gievská, S., Sibert, J.: Empirical Validation of a Computer-Mediated Coordination of Interruption. In: CHISIG - OZCHI 2004: Supporting Community Interaction, Wollongong, Australia, pp. 271–280 (2004)
6. Horvitz, E., Kadie, C., Paek, T., Hovel, D.: Models of Attention in Computing and Communication from Principles to Applications. Communications of the ACM 46, 52–59 (2003)
7. Oliver, N., Horvitz, E.: A comparison of HMMs and dynamic bayesian networks for recognizing office activities. In: Ardissono, L., Brna, P., Mitrović, A. (eds.) UM 2005. LNCS (LNAI), vol. 3538, pp. 199–209. Springer, Heidelberg (2005)
8. Jambon, F.: Formal Modelling of Task Interruptions. In: CHI 1996, pp. 45–46 (1996)
9. Kim, Y., Baylor, A.: Pedagogical Agents as Learning Companions: The Role of Agent Competency and Type of Interaction. Association for Educational Communications and Technology 54, 223–243 (2006)
10. Yin, B., Chen, F.: Towards Automatic Cognitive Load Measurement from Speech Analysis. In: Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4550, pp. 1011–1020. Springer, Heidelberg (2007)
11. Sykes, E.R.: Determining the Effectiveness of the 3D Alice Programming Environment at the Computer Science I Level. Journal of Educational Computing Research 36 (2007)
12. Sykes, E.R.: Developmental Process Model for the Java Intelligent Tutoring System. Journal of Interactive Learning Research 18 (2007)
13. Sykes, E.R., Franek, F.: A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java. International Journal of Computers and Applications 1, 35–44 (2004)
14. Horvitz, E., Koch, P., Apacible, J.: BusyBody: Creating and Fielding Personalized Models of the Cost of Interruption. In: CSCW 2004, vol. 6, pp. 507–510. ACM, Chicago (2004)
15. Yu, K., Wang, H., Wu, X.: A Parallel Algorithm for Learning Bayesian Networks. In: Zhou, Z.-H., Li, H., Yang, Q. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4426, pp. 1055–1063. Springer, Heidelberg (2007)
16. O'Reilly, R.C., Munakata, Y.: Computational Explorations in Cognitive Neuroscience. MIT Press, London (2000)
17. Felder, R.M., Silverman, L.K.: Learning and Teaching Styles in Engineering Education. Engineering Education 78, 674–681 (1988)
18. GazeTracker Software Guide (2008)

19. Namasivayam, V., Prasanna, V.: Scalable Parallel Implementation of Exact Inference in Bayesian Networks. In: 12th International Conference on Parallel and Distributed Systems, vol. 1, pp. 143–150. IEEE, Los Alamitos (2006)
20. Iqbal, S., Bailey, B.: Leveraging Characteristics of Task Structure to Predict the Cost of Interruption. In: CHI 2006, pp. 741–750. ACM, Montreal (2006)
21. Iqbal, S., Horvitz, E.: Disruption and Recovery of Computing Tasks: Field Study, Analysis, and Directions. In: CHI 2007. ACM, San Jose (2007)
22. Garcia, J.C.F., Mendez, J.J.S.: A Comparison of ANFIS, ANN and DBR systems on volatile Time Series Identification. In: Annual Meeting of the North American Fuzzy Information Processing Society, pp. 319–324. IEEE, Los Alamitos (2007)
23. Gharaviri, A., Dehghan, F., Teshnelab, M., Abrishami, H.M.: Comparison of Neural Networks, ANFIS, and SVM Classifiers for PVC Arrhythmia Detection. In: Proceedings of the Seventh International Conference on Machine Learning and Cybernetics, vol. 2, pp. 750–755. IEEE, Kunming (2008)
24. Jang, J.S.R.: ANFIS: Adaptive-Network-Based Fuzzy Inference System. *IEEE Trans. Systems, Man, Cybernetics* 23, 665–685 (1993)

SCTP, XTP and TCP as Transport Protocols for High Performance Computing on Multi-cluster Grid Environments

Diogo R. Viegas¹, R.P. Mendonça¹, Mario A.R. Dantas¹, and Michael A. Bauer²

¹ Post-Graduate Program in Computer Science (PPGCC),

Universidade Federal de Santa Catarina (UFSC),

Campus Trindade, Florianopolis, 88040 900, Brazil

{diogo, rodrigop, mario}@inf.ufsc.br

² Department of Computer Science,

University of Western Ontario,

London, Ontario, N6A 5B7, Canada

bauer@csd.uwo.ca

Abstract. Multi-cores and multi-processors loosely coupled architectures are interesting commodity off-the-shelf architectures with which to build multi-cluster configurations as grid environments. However, such grid environments face challenges that must be circumvented. Examples include the heterogeneity of computational resources (e.g. operating systems, processors), of different programming paradigms and of network protocols. Transport protocols are especially important, because they can directly impact the execution of distributed engineering and scientific applications. In this paper, we present an empirical study of the SCTP, XTP and TCP as transport protocols with which to support applications in multi-cluster configurations. The environment includes several networks of workstations (NOWs) interconnected via a wide area network. Our experiments consider throughput and latency parameters against number of flows; the results show performance differences from the three transport protocols over a real distributed production configurations.

Keywords: Multi-clusters, high-performance protocols, SCTP, XTP, grid configurations.

1 Introduction

In many organizations, the use of loosely coupled configurations of multi-cores and multi-processors in high performance computing (HPC) configurations are known to enhance applications' performance. Such distributed configurations are usually referred to as a network of workstations (NOWs). However, as observed [1], such configurations are not well suited to a number of applications. This observation is magnified when a wide distributed configuration (e.g., multi-clusters as a grid environment) is considered to form an HPC infrastructure. In this case, not only the machines' heterogeneity, as stated in [2, 3, 4], but also the programming approach [5, 6] are potentially problematic.

Network technologies—both hardware and software—include obstacles to be overcome. A classical issue is conventional computer network’s transport protocol, which usually adopt a heavyweight or best-effort paradigm (e.g., TCP and UDP) to support the execution of applications.

As a result, the literature shows two different research approaches attempting to solve the problem. The first approach is characterized by general research concerning how to improve the efficiency of existing protocols [7, 8], or even new proposed protocols [9, 10, 11]. The other approach is characterized by its focus on specific aspects related to high performance computing [12, 13, 14].

Here, we present an experimental study of the use of the SCTP, XTP and TCP transport protocols to support applications over multi-cluster configurations. Our experimental environment, unlike work presented in [12, 13, 14], does not have a laboratory boundary. Instead, our experiments were realized in a large organization in a day-to-day basis of routers, switches, and firewalls with unpredictable workloads and latencies.

This paper is organized as follows. Relevant aspects of the SCTP, XTP and TCP transport protocols are presented in Section 2. Section 3 illustrates some related investigations. In Section 4, we discuss an experimental scenario used to conduct this research. In addition, experimental results are presented. Finally, some conclusions and suggestions for future work are discussed in Section 5.

2 Fundamentals of SCTP, XTP and TCP

In the TCP/IP architecture, the transport protocol layer is responsible for providing services to the application layer. Example services include reliable messages, reliable data (sequence of octets), and both acknowledged and unacknowledged datagrams. The two main transport services of this architecture are TCP and UDP, respectively. The TCP is characterized as an acknowledged transport of octets’ sequences. By contrast, the UDP is a non-acknowledged transport of messages [15, 16]. The TCP/IP architecture did not originally offer a non-fragmented acknowledged message service [15, 16]. As a result, in this section we contrast theoretical differences between the SCTP, XTP and TCP transport protocols. Our goal is to show how each protocol’s main mechanisms are used to provide an efficient communication to the application layer.

2.1 Stream Control Transmission Protocol (SCTP)

The Stream Control Transmission Protocol (SCTP) [17, 18] is a message-oriented reliable transport protocol that operates on the top of an unreliable connectionless packet service, such as IP. It was originally developed to carry telephony signaling messages over IP networks for telecommunications systems. Its evolution as a general purpose protocol was natural due to the absence of necessary features in the existing TCP/IP architecture protocols. In addition to the facilities inside the TCP, the SCTP provides a low latency connection avoiding the head-of-line problem, well known in TCP. Because of low-latency connections, the protocol can be used in heavily congested networks. This characteristic is essential for NOWs as multi-cluster distributed

configurations. Such environments rely on networks (private and public), where congestions are usually common.

The SCTP is a session-oriented protocol that creates n multiple independent logical stream endpoints. The protocol enhances the throughput by allowing parallel connections without the necessary use of multiple socket connections.

In contrast to the TCP byte-oriented acknowledge-retransmission model, SCTP adopts an approach in which messages are fragments of chunks. Transport Sequence Number (TSN), Stream Identifier (SI), and Stream Sequence Number (SSN) are the triple that allows the fragmentation and re-assembly of messages in the multi-stream architecture. Figure 1 highlights the main fields of the previous described SCTP package.

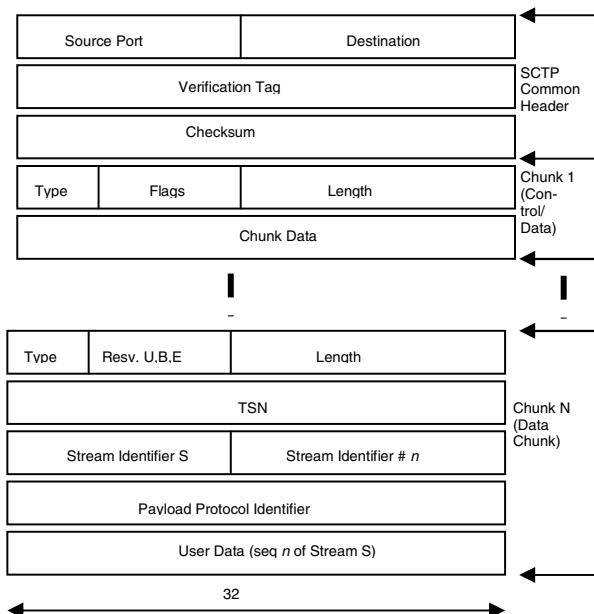


Fig. 1. SCTP package format [17]

Like the sequence number from the TCP protocol, the TSN is responsible for acknowledgment and retransmission mechanisms. The SI identifies the stream to which the user's data belong. Finally, the SSN identifies messages within a stream.

The SCTP also provides multi-homing, i.e., in a SCTP association, each endpoint can be bound to multiple interfaces. This facility implies that an association can build all paths among interfaces. Therefore, if a failure occurs, the SCTP protocol can automatically provide another fail-over path, without the interaction of any application.

Finally, an important characteristic of the SCTP is its four-way handshake and cookie mechanism used to establish associations. It avoids denial-of-service attacks (e.g., SYN flooding), because the SCTP listening end-point allocates resources only after receiving a cookie echo request from the client. In [19] a list of SCTP implementations is presented.

2.2 Xpress Transport Protocol (XTP)

The Xpress Transport Protocol (XTP) is a transport layer protocol that offers new features compared to conventional transport protocols such as TCP and UDP. This protocol was primarily conceived to be used in specialized high performance networks, such as Safenet [20]. Several functions of the XTP protocol were designed to provide more flexibility to the execution of distributed applications, examples are [21]:

- Multicast and management of multicast groups;
- Support to priority;
- Transmission control rate;
- Selective retransmission.

Additional facilities of the XTP are error control, flow control, and rate control. Instead of separate protocols for each type of communication, XTP controls packet exchange patterns to produce different models, e.g., reliable datagrams, transactions, unreliable streams, and reliable multicast connections. The protocol does not use congestion-avoidance algorithms.

XTPSandia [22] is an example of XTP implementation [23] which uses concepts from the object-oriented approach. The package uses a Meta transport library called MTL. MTL is a collection of classes written in C++ from where it is possible to obtain the transport protocol implementation. The idea behind the XTPSandia is a daemon executing in the user space. It is also interesting to mention that the XTPSandia has a feature that allows an ordinary user to customize the protocol to a large number of different operating systems such as Linux, Solaris, and AIX.

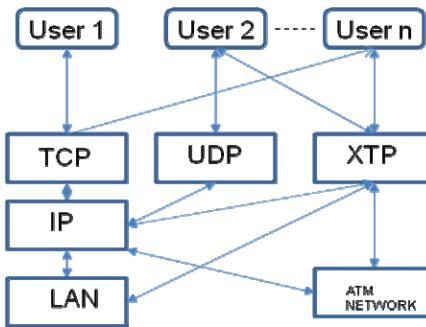
Figure 2 shows possible XTP interconnections options. From Figure 2, one can see that network-in-box technologies (e.g., Infiniband, Myrinet) can adopt the XTP as a transport to enhance the communication among processing nodes.

3 Related Works

Several ongoing investigations focus on the efficient use of computer and communication networks (e.g., [8,9,13,14]). There are broadly two different schools researching this problem. One group considers a general use of the network (e.g., [10],[11]). In other words, it does not focus on a specific application. The approach's objective is to provide a better throughput with a low latency for different applications. This could be a serious drawback for a distributed and parallel application if no QoS (Quality of Service) policy is adopted.

By contrast, the second school focuses on distributed and/or parallel applications (e.g., [12, 14]). These efforts are interesting for scientific and engineering applications because they often rely on local clusters for execution. However, proposals are in initial stages and proposed solutions are for specific infrastructures.

Therefore, when considering multi-cluster configurations as grid environments with which to execute distributed applications, it would be better to have a hybrid investigation, i.e., the use of a general purpose new protocol on a commodity distributed configuration for the execution of distributed applications.

**Fig. 2.** Protocol interconnections options [23]

In this direction, recent research, presented in [24], [25] and [26], highlights the importance of adopting the SCTP protocol, standardized by the IETF, to be used in IP based applications. Therefore, investigations presented in [24, 25, 26] have similar interest as the present work. However, unlike [24]—where the authors affirm that they have tested in a small cluster with a limited scalability—our experiments were considered in a real scenario of a wide distributed network (as illustrated in Fig. 4). In [25] a similar environment is reported, where a local cluster of ten nodes were used, but the research only adopted the SCTP protocol. In [26], twenty-four homogeneous

Table 1. Characteristics of SCTP, TCP and XTP

Characteristics	SCTP	TCP	XTP
Reliable	Yes	Yes	Yes
Connection oriented	Yes	Yes	Yes
Unit of transmission	Msg	Octet	Msg
Flow control	Yes	Yes	No
Congestion control	Yes	Yes	Yes
Failure detection	Yes	No	Yes
Data distribution	P-order	Order	Order
Multi-homing	Yes	No	No
Multiple streams	Yes	No	No
Ordered data delivery	Yes	Yes	Yes
Unordered data delivery	Yes	No	Yes
Selective ACK	Yes	Optional	Yes
Checksum pseudo-header	No	Yes	Yes
Half-closed connections	Yes	Yes	Yes
SYN flooding attacks protection	Yes	No	No

nodes were used as a local cluster using Fast-Ethernet network cards and a Gigabit interconnection network. It was restricted to SCTP and TCP protocols.

In this work, we considered the SCTP, XTP and TCP transport protocols, and also a real distributed area network to execute some tests. Our contribution differs from other researchers, especially because it considers multi-clusters as a grid configurations. Table 1 shows characteristics related to SCTP, TCP and XTP.

4 Experimental Scenario and Results

In this section, we describe specific characteristics from the grid environment, represented by four multi-cluster configurations, employed for our experiments. In addition, we present empirical results utilizing the SCTP, XTP and TCP transport protocols executing in the grid environment.

Multi-cluster grid configurations are usually characterized by the use of resources that are physically distributed and under different management policies. In other words, resources are distributed over a wide area network from different organizations, creating a *virtual organization* (VO). Figure 3 exemplifies a multi-cluster grid configuration, which is formed by three different clusters where grid resources can be utilized.

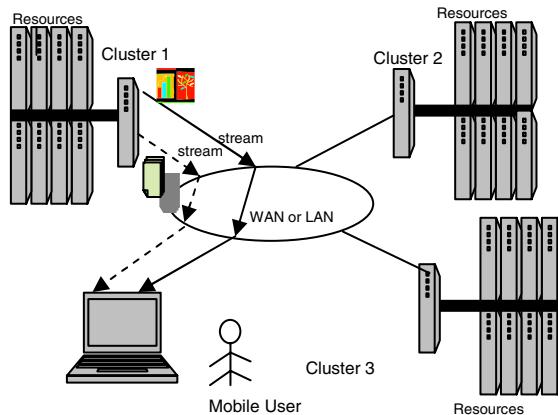


Fig. 3. Multi-cluster grid configuration example

Figure 3 also illustrates a mobile user accessing a multimedia application that returns from Cluster 1, a text and an image. Considering this example, would be desirable if such a multi-streaming application could be supported by the transport protocol. In addition, it would also be desirable to use the same service when any congestion occurs in the network. In other words, it is expected that the throughput and latency could be inside an acceptable threshold.

Our tests are based on measurements from a real configuration in a large organization. As expected, several networks co-existed in the organization (examples were VoIP, PSTN, GPRS, and IP). Figure 4 illustrates the test bed with all available resources. The VO of our grid was formed by four multi-cluster configurations (i.e. *Alfa*, *Beta*, *Gama* and *Delta*). The multi-cluster configurations, shown in Figure 4, have a similar function to those from Figure 3. However, a differential aspect is related to unpredictable workload and latency from the networking. In other words, the configuration operates in a day-to-day basis with routers, switches, and firewalls with unpredictable workloads and latencies.

Demonstration applications used in the test bed were designed and implemented by our research group, and can be found in [27]. The applications allow us to change message sizes, number of flows for SCTP and bind addresses. These applications were developed to verify all characteristics of the SCTP, XTP and TCP transport protocols.

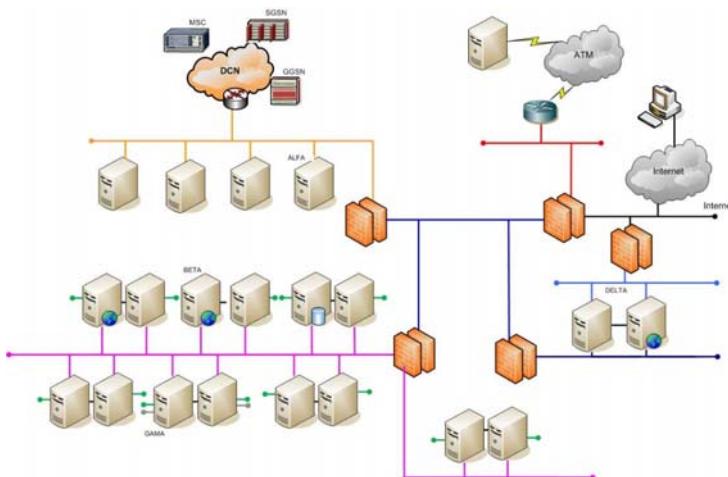


Fig. 4. Virtual organization (VO) multi-cluster configurations

Figures 5 and 6 illustrate results from the inter-communication of machines within the *Alfa* and *Delta* clusters. A significant aspect for inter-cluster communications is the number of flows between these environments. Multi-stream applications can benefit from a pipeline with several sub-channels. This is a special feature found in new protocols, such as SCTP and XTP. Figure 5 shows the throughput and flow performance of the three tested protocols. In theory, it was expected that SCTP would perform better, since multiple flows are an inherent characteristic of SCTP. However, the security facility has a drawback effect on this protocol in these experiments especially because there were two firewalls between the two clusters. We discovered that the CRC-32c checksum calculation and packet splitting penalized the SCTP and XTP protocols. Without using these features we obtained a similar performance numbers between the three protocols. Nevertheless, we understand that all comparisons should consider common aspects from these protocols

As shown in Figure 6, as the number of flows increased, the latency of the SCTP decreased. However, the XTP and TCPM had the same latency performance as the number of flows increased. We used the enhanced TCPM (TCP Maintenance and Minor Extensions) protocol, when standard TCP did not have a feature that could be contrasted to the SCTP and XTP protocols.

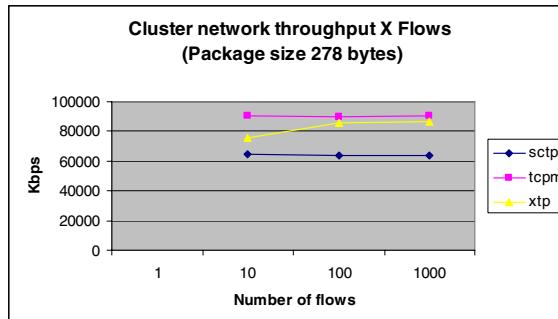


Fig. 5. Cluster network throughput x flow 278 bytes

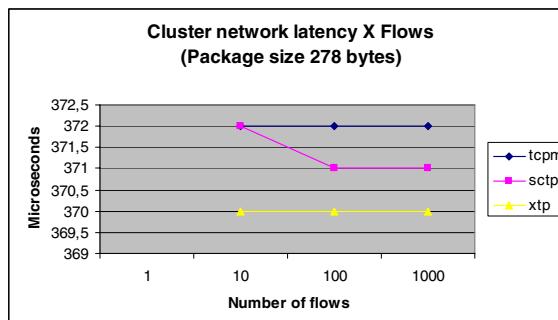


Fig. 6. Cluster network latency x flow 278 bytes

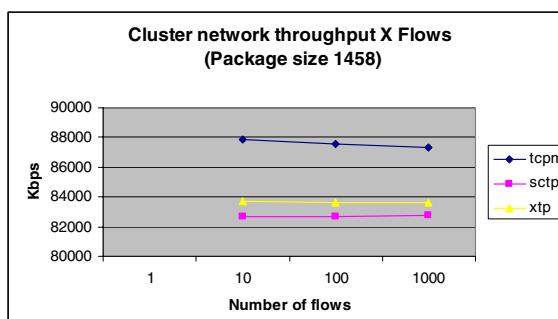


Fig. 7. Cluster network throughput x flow 1458 bytes

Experimental results shown in Figures 7 and 8 represent the same inter-cluster communication pattern infrastructure from Figures 5 and 6, but with message sizes equal to 1458 bytes. The throughput of TCPM decreased when the number of concurrent flows increased, as illustrated in Figure 7. However, XTP and SCTP maintained their performance pattern. The graph of latency in Figure 8 shows that XTP had lower performance for all flows, whereas the TCPM and SCTP had the same continuous latency for the threshold between ten and a thousand flows (because of that the SCTP is shadowing the TCPM line).

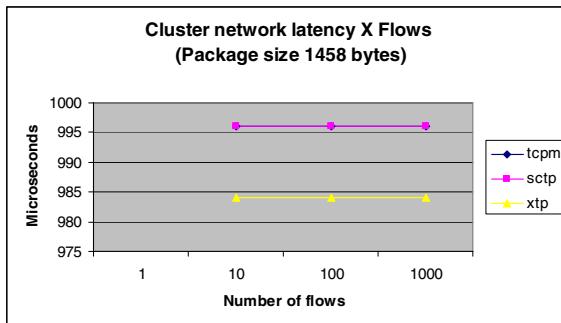


Fig. 8. Cluster network latency x flow 1458 bytes

5 Conclusions and Future Work

In this paper we have compared the SCTP, XTP and TCP transport protocols. Unlike other studies, we considered a real VO scenario formed by a multi-cluster configuration.

Our experiments indicate that SCTP has the drawback of security aspects in terms of throughput against number of flows. However, latency decreases while the communication links increase. These characteristics are interesting because it is possible to consider less security to achieve more improved throughput levels. On the one hand, SCTP could be seen as a secure transport with a reasonable latency to execute distributed and parallel applications. On the other hand, TCP and XTP represent interesting options in terms of throughput and latency for environment where applications do not require multi-homing, fail-over or deny-of service protection.

For future work, we plan to integrate to the present research into the work presented in [28]. This work aims to provide a dynamic resource-matching feature for multi-clusters. The use of a lightweight protocol, together with this resource approach, can be an interesting contribution. The objective is to provide to engineering and scientific users with an enhanced multi-cluster grid environment in which to execute complex applications.

References

1. Strong, J.: HCW Panel: Programming heterogeneous systems - Less pain! Better performance! In: IEEE International Symposium Parallel and Distributed Processing - IPDPS (2006)

2. Mendonça, R.P., Dantas, M.A.R.: A Study of Adaptive Co-scheduling Approach for an Opportunistic Software Environment to Execute in Multi-core and Multi-Processor Configurations. In: Proceedings of the 11th IEEE International Conference on Computational Science and Engineering, CSE 2008, pp. 41–47 (2008)
3. El-Moursy, A., Garg, R., Albonesi, D.H., Dwarkadas, S.: Compatible phase co-scheduling on the CMP of multi-threaded processors. In: IEEE International Symposium Parallel & Distributed Processing - IPDPS (2006)
4. Pinto, L.C., Tomazella, L.H., Dantas, M.A.R.: An Experimental Study on How to Build Efficient Multi-core Clusters for High Performance Computing. In: Proceedings of the IEEE CSE 2008, pp. 33–40 (2008)
5. Pourreza, H., Graham, P.: On the Programming Impact of Multi-Core, Multi-Processor Nodes in MPI Clusters. In: Proceedings of the 21th International Symposium on High Performance Computing Systems and Applications - HPCS, p. 1 (2007)
6. Cvetanovic, Z.: The Effects of Problem Partitioning, Allocation and Granularity on Performance of Multiple-Processor Systems. *IEEE Trans. On Computers* 36(4), 421–432 (1987)
7. Hassan, M., Jain, R.: High Performance TCP/IP Networking: Concepts, Issues, and Solutions. Prentice Hall, Englewood Cliffs (2004)
8. Chakravorty, R., Katti, S., Crowcroft, J., Pratt, I.: Flow Aggregation for Enhanced TCP over Wide-Area Wireless. In: Proceedings of the IEEE 22nd Infocom, pp. 1754–1764 (2003)
9. Caro Jr., A.L., Iyengar, J.R., Amer, P.D., Ladha, S., Heinz II, G.J., Shah, K.C.: SCTP: A Proposed Standard for Robust Internet Data Transport. *IEEE Computer* 36(11), 56–63 (2003)
10. Fu, S., Atiquzzaman, M.: SCTP: state of the art in research, products, and technical challenges. *IEEE Communication Magazine* 42(4), 64–76 (2004)
11. Dantas, M.A.R., Jardini, G.: Performance Evaluation of XTP and TCP Transport Protocols for Reliable Multicast Communications. In: Proceedings of HPCN Europe, pp. 591–594 (2001)
12. Díaz, A.F., Ortega, J., Cañas, A., Fernández, F.J., Prieto, A.: The Lightweight Protocol CLIC: Performance of an MPI implementation on CLIC. In: Proceedings of the IEEE International Conference on Cluster Computing, CLUSTER 2001, pp. 391–398 (2001)
13. Kamal, H., Penoff, B., Wagner, A.: SCTP versus TCP for MPI. In: Proceeding of ACM/IEEE Supercomputing Conference, p. 30 (2005)
14. Jin, H., Zhang, M., Tan, P., Chen, H., Xu, L.: Lightweight Real-Time Network Communication Protocol for Commodity Cluster Systems. In: Yang, L.T., Amamiya, M., Liu, Z., Guo, M., Rammig, F.J. (eds.) EUC 2005. LNCS, vol. 3824, pp. 1075–1084. Springer, Heidelberg (2005)
15. Tanenbaum, A.S.: Computer Networks, 4th edn. Prentice-Hall, Englewood Cliffs (2002)
16. Stevens, R.W.: TCP/IP Illustrated. The Protocols, vol. 1. Addison-Wesley, Reading (1994)
17. RFC 2960, SCTP, <http://www.ietf.org/rfc/rfc2960.txt>
18. Stewart, R.R., Xie, Q.: Stream Control Transmission Protocol (SCTP): A Reference Guide. Addison-Wesley, Reading (2002)
19. SCTP (December 2008), <http://www.sctp.org/implementations.html>
20. Cohn, M.: A Lightweight Transfer Protocol for the U.S. Navy Safenet Local Area Network Standard. In: Proceedings of the 13th Conference on Local Computer Networks, pp. 151–156 (1988)
21. Simoncic, R., Weaver, A.C., Colvin, M.A.: Experience with the Xpress transfer protocol. In: Proceedings of the 15th Conference on Local Computer Networks, pp. 123–131 (1990)

22. Timothy, W., Michael, S., Lewis, J., Cline, R.E.: XTP as a Transport Protocol for Distributed Parallel Processing. In: Proceedings of the High-Speed Networking Symposium on Usenix, p. 6 (1994)
23. XTP,
<http://users.enics.concordia.ca/~bill/hspl/xtplinux/vincelofaso.html>
24. Zarrelli, R., Petrone, M., Iannaccio, A.: Enabling PVM to exploit the SCTP protocol. Journal of Parallel and Distributed Computing 66(11), 1472–1479 (2006)
25. Penoff, B., Tsai, M., Iyengar, J., Wagner, A.: Using CMT in SCTP-Based MPI to Exploit Multiple Interfaces in Cluster Nodes. In: Proceedings of the EuroPVM/MPI, pp. 204–212 (2007)
26. Kozlovszky, M., Berceli, T., Kutor, L.: Analysis of SCTP and TCP based communication in high-speed clusters. Nuclear Instruments and Methods in Physics Research Section A 559(1), 85–89 (2006)
27. Pfutzenreuter, E.: Aplicabilidade e desempenho do protocolo de transporte SCTP, Msc Thesis, Informatics and Statistic Department (INE), Federal University of Santa Catarina (UFSC), <http://www.ppgcc.inf.ufsc.br/>
28. Fereira, D.J., Silva, A.P.C., Dantas, M.A.R., Qin, J., Bauer, M.A.: Dynamic Resource Matching for Multi-Clusters Based on an Ontology-Fuzzy Approach. In: High Performance Computing Symposium, Kingston, Canada (2009)

Dynamic Resource Matching for Multi-clusters Based on an Ontology-Fuzzy Approach

Denise Janson¹, Alexandre P.C. da Silva¹, M.A.R. Dantas¹,
Jinhui Qin², and Michael A. Bauer²

¹ Post-Graduate Program in Computer Science (PPGCC),
Universidade Federal de Santa Catarina (UFSC),
Campus Trindade, Florianopolis, 88040 900, Brazil

{ded,parra,mario}@inf.ufsc.br

² Department of Computer Science,
University of Western Ontario,
London, Ontario, N6A 5B7, Canada
{jhqin,bauer}@csd.uwo.ca

Abstract. One key aspect for the successful utilization of grid environments is how to efficiently schedule distributed and parallel applications to these configurations. It is also desirable to make the tlymatching operation of available resources as transparent as possible to the user. These aspects are especially important for grid environments formed by heterogeneous multi-cluster machines. In this paper, we present an approach that considers both computer resources and communication links. The approach is based on a combination of ontology and fuzzy logic. The ontology paradigm is employed as a standard interface to accept users's requirements for desired resources. The fuzzy logic algorithms are used to compute parameters for matching based on dynamically monitored values of processor usage and communication. Experimental results indicate that the proposed approach is successful in terms of gathering dynamically more appropriate distributed resources and communication links in multi-cluster environments.

Keywords: Resource matching, meta-scheduling, ontology, fuzzy, multi-cluster.

1 Introduction

Cluster environments are common configurations used in many organizations to provide high performance computing (HPC) resources for a variety of applications. Multi-clusters, if well orchestrated as a grid environment, can represent significant computational power for the execution of several classes of applications, including parallel jobs. A grid environment, composed of multi-cluster configurations, can be considered in a private or in multi-domain organizations. In other words, this meta-computing environment can be employed as a HPC facility inside a specific organization, or among different organizations, creating in both cases the concept of a virtual organization (VO).

One important challenge in this type of grid environment is how to reach an efficient allocation of resources to local and remote applications. Inside a cluster configuration, a local resource management system (RMS) will naturally provide high priority to local applications. Nowadays, only a few RMSs can support the facility of advance reservation (AR). On the other hand, a RMS that supports AR (e.g. MAUI/MOAB [1], PBS Professional [2], LSF [3] and Loadlever [4]) utilizes a private interface format for the execution of this advance reservation.

RMSs are commonly classified into two classes. Those that no AR facility exists, thus its scheduler provides high priority to local jobs in multi-cluster environments, when compared to remote jobs. The second class of RMS is characterized by a built-in AR function. Thus, remote jobs could also have a high priority comparable to local jobs.

RMS packages that have a built-in AR component are broadly referred to *meta-schedulers*. *Meta-schedulers* are characterized by receiving users's requests and then scheduling an application to a local cluster system. Therefore, *meta-schedulers* interfers on local scheduling to orchestrate multi-cluster configurations, because these components can change both local and remote queues. Also, this AR functionality from a *meta-scheduler* needs efficient design of co-reservation and co-allocation modules.

Thus, a *meta-scheduler* proposal with an efficient algorithm for co-reservation and co-allocation is a key element in grid environments, especially those formed by multi-cluster configurations. A number of *meta-scheduler* projects for grid environments have already been proposed with different characteristics. Examples of grid *meta-schedulers* are Calana [5], Deco [6] Gridway [7], GridARS [8] and Viola [9].

Another import aspect of *meta-schedulers* is the challenge of dealing with different heterogeneous local resource management systems in multi-cluster configurations. This problem occurs because of the utilization of private heterogeneous interfaces from AR modules. Only a few projects, such as Community Scheduler Framework (CSF) [10], promise to have a standard interface for information exchange between RMSs. However, these efforts are far from the final goal [11].

The first step in building a *meta-scheduler* is to design and implement an AR module. In this paper, we present an AR module represented by standard interface that implements a dynamic resource and communication link matching approach. In other words, the present proposal considers computer resources and communication links characteristics, gathered from nodes of multi-cluster configurations to provide a standard grid interface. The system employs ontology and fuzzy logic algorithms for interactions between application programmers and the system. The ontology paradigm is employed as a standard representation of available resources that can be matched by the users's choice to support the execution of their applications. Besides, the fuzzy logic algorithms are used to control and to set dynamically the matching from users's requests and available resources and communication links in a multi-cluster configuration.

The remainder of this paper is organized as follows. In section 2, a brief overview of related work is provided. The proposed approach, including components and interactions, is described in section 3. Section 4 shows the experimental

environment and some results. Finally, in section 5 we present some conclusions and future directions related to this research.

2 Related Research Work

The research work presented in [12], shows an ontology oriented framework which targets the management of Virtual Organizations (VOs). The authors claim that the framework, with a generic ontology, improved the processing and storage requirements submitted to the environment. The system has three layers. The infrastructure is the lowest layer, which is hidden from users by an intermediate semantic level. The monitoring and scheduling functions are two components that are being improved to reach the desirable transparencies. One example of a transparency goal is the resource collaboration usage between VOs.

In [13], a framework is presented that is called the *Semantically - Enhanced Resource Allocator*, that has the goal to improve the scheduling and resource allocation. An objective of this work is also to provide a SLA (Service Level Agreement) facility in each service provider for the configuration. Interactions among nodes of the grid are regulated through the execution of the client manager, application manager, semantic metadata repository and semantic scheduler modules.

A resource matching approach based on multiple ontologies can be found in [14]. A reference and query ontology was conceived to integrate different ontologies from different VOs. The system provides an ordinary user a friendly interface to select some parameters such as operating system version, processor architecture, processor clock speed and amount of memory. This proposal does a static integration of multi-cluster environments, in addition to the interaction with end-users.

A job co-allocation proposal is presented in [15]. This research addresses the allocation of large jobs into multi-cluster configurations. This contribution considers a job splitting function which takes into consideration the availability of processors and communication link bandwidth between multi-cluster nodes. A fuzzy logic approach is used to control the job splitting operation and bandwidth usage. Policies adopted for the job splitting and link allocation are shown in [16].

3 Proposed Approach

The design and implementation of our proposal represents an extension of the research in [14] and [15].

3.1 Design Aspects

The resource matching proposed in [14], which utilizes the ontology approach, is illustrated in Fig. 1. In this environment from an integration portal it is possible to access the reference ontology, query ontology and rules. It is also possible to

store relations and request data from information providers. In addition, from the query interface it is possible to access a matching service which employs the Jena Generic Rule Reasoner [17]. In other words, an application programmer can search for an environment to execute his or her application through the use of this system. As presented in [14] this proposal can enlarge queries providing more information of the available grid environment. As a result, during a query a configuration can be built to execute an application, considering multi-cluster configurations.

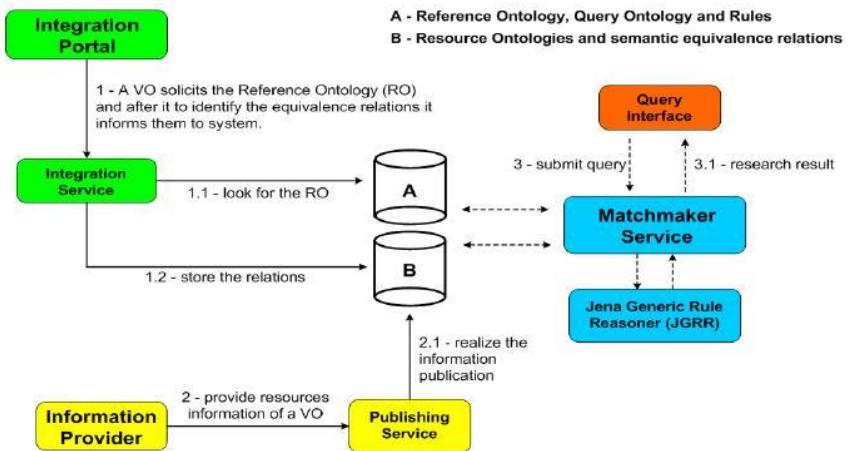


Fig. 1. The resource matching based on ontology [14]

All operations realized in [14] consider only static information; no consideration is taken related to information change. This is an interesting point to be improved, because in multi-cluster resources availability can change frequently. In addition, there is no consideration of the load on communication links among multi-cluster. A natural enhancement for this configuration, to improve the execution of distributed and parallel applications, is to dynamically gather computer resources and communication link metrics.

In this paper, the research of [14] was continued in order to find an effective solution to this gap. Utilizing the fuzzy logic algorithms proposed in [15], it is possible to monitor the environment, get threshold information from cluster resources and communication links. As a result, it is expected that the matchmaker service could reach better multi-cluster scenarios since a more precise set of information could be translated into better configurations to execute applications.

Our proposal modifies the original resource matching shown in Fig. II, sending the submitted query to a new *Allocator* module that interfaces with the fuzzy module from [15]. The idea behind this change is to collect fresh information related to computer resources and communication links utilization. The result

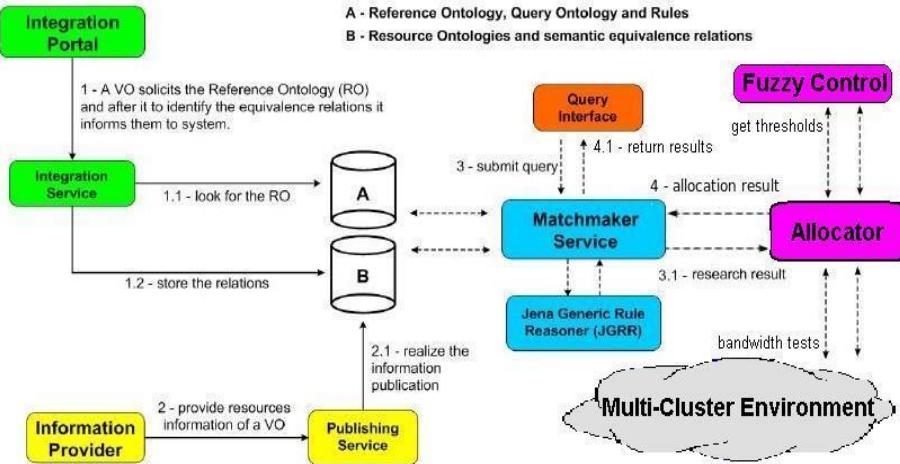


Fig. 2. The new proposed architecture

from this new query is called as *allocation result*. Fig. 2 shows the new design of the environment after inserting the new modules.

3.2 Implementation Characteristics

The transparency in the proposed approach can be achieved by adding the fuzzy algorithms inside some functions of the Matchmaker Service in the new Allocator module (Fig. 2). The original Matchmaker (Fig. 1) selects resources based on a threshold provided by an end-user. A threshold example could be seen as a number of required processors to execute an application. If this number is not reached, the Matchmaker sends a warning to the end-user saying that it is not possible to execute the application.

In the proposed environment, a similar query needing processing nodes is sent to the new module that is able to split jobs across existing multi-cluster. The new proposal is able to check dynamically the multi-cluster configuration required. Thus, if new computer resources were available within the requirements of the query, it will immediately be considered.

The original code from the research presented in [14] was developed in JAVA. The Matchmaker interface was enhanced aiming to receive more information from the new Allocator component. The Allocator module (Fig. 2), which was implemented in C, is capable of calculating communication links latencies between clusters in addition to providing the number of available nodes on a cluster.

The Allocator component is capable of assigning jobs to different multi-cluster communication links. This feature is realized with the help of the fuzzy logic algorithms presented in [16]. This part of the code, implemented in C++, is responsible for calculating the threshold (δ) which controls a job splitting and utilization level of a communication link (λ). The utilization of communication

link divided by its maximum capacity must be less than λ , otherwise the link is saturated. Aiming to reduce the communication among clusters, a portion of job size (S) versus the threshold (δ) is required to be executed inside a unique cluster. This approach allows a better load balancing and also it considers saturated communication links. The ontology components were built utilizing Protégé 3.3.1

4 Experimental Environment and Results

In this section we present our experimental environment and results from some case studies using our proposal. In order to show how the proposal represents an enhanced configuration, we start our experiments considering characteristics from the original environment, and then employing new features from the new system. The new system is illustrated in Fig. 2

The multi-cluster environment was simulated using the SimGrid [18] software package.

The goal of the SimGrid project is to facilitate research in the area of distributed and parallel application scheduling on distributed computing platforms, ranging from a simple network of workstations to large computational grids. The SimGrid toolkit provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. This characteristic of the package is interesting since it enables simulation of multi-cluster configurations. Table 1 shows characteristics from the two virtual organizations that were created for our experiments.

Table 1. Multi-cluster configuration

	VO-A	processors	VO-B	processors
Cluster_01	7		Cluster_1	4
Cluster_02	5		Cluster_2	4
Cluster_03	3		-	-

Table 2. The ontology set utilized for the experiments

Request's Concepts	VO-B's Concepts	VO-A's Concepts
min_number_of_cpus	[numero_cpus]	[number_of_cpus]
min_virtual_memory_available	[swap_livre_MB]	[free_virtual_memory]
min_clock_speed	[velocidade_cpu]	[max_clock_speed]
owner	[nome_conta]	[login]
number_of_cpus	[numero_cpus]	[number_of_cpus]
min_physical_memory_available	[memoria_livre_MB]	[free_physical_memory]
min_free_disk_space	[espaco_disco_livre_GB]	[available_space]
os_type	[SO, SistemaOperacional]	[os_type]
max_load_cpu_15min		[percentage_load_15min]
max_load_cpu_5min		[percentage_load_5min]
max_load_cpu_1min		[percentage_load_1min]

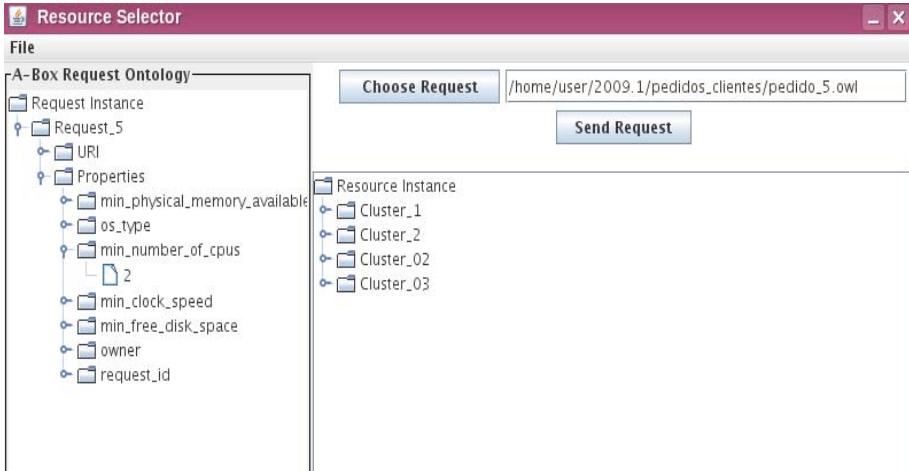


Fig. 3. Quering 2 processors. Old system without dynamic information.

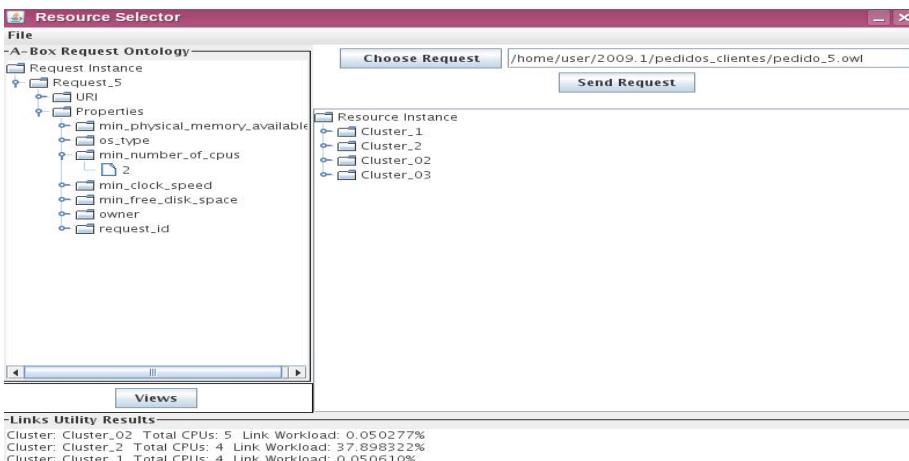


Fig. 4. Quering 2 processors. New system with dynamic information available.

After creating the multi-cluster configuration we created a number of ordinary concepts to be utilized in both virtual organizations for requesting resources. Table 2 presents how these request concepts are understood in virtual organizations A and B. Virtual organization A represents a Canadian environment, whereas virtual B is a Brazilian configuration.

The first test was characterized by a query where a user required a certain minimum amount of physical memory, operating system type of UNIX and number of processors equal to 2. Fig. 3 shows the result from this first query. This experiment was exactly as it was reported in the research shown in [14]. The

idea of this experiment was to show that the contribution acts similarly to the original proposal.

Fig. 4 shows the second test, where the same query was executed under the new system. In addition to the available environment, this figure presents numbers related to the number of processors and workload.

The third experiment represents a query as it was in original environment, where a user requires 6 processors. Fig. 5 shows the result for this query. As expected, the system replies that no match was found. This answer is based upon the static information available in the original system.

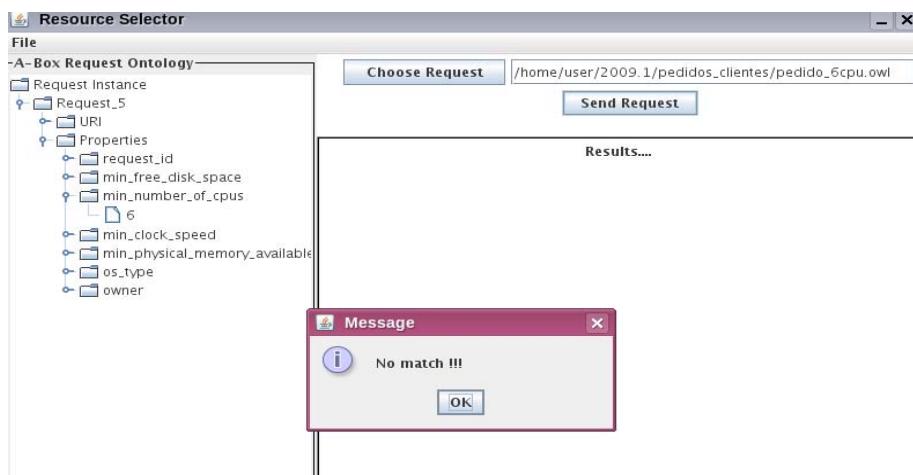


Fig. 5. Quering 6 processors. Old system without dynamic information.

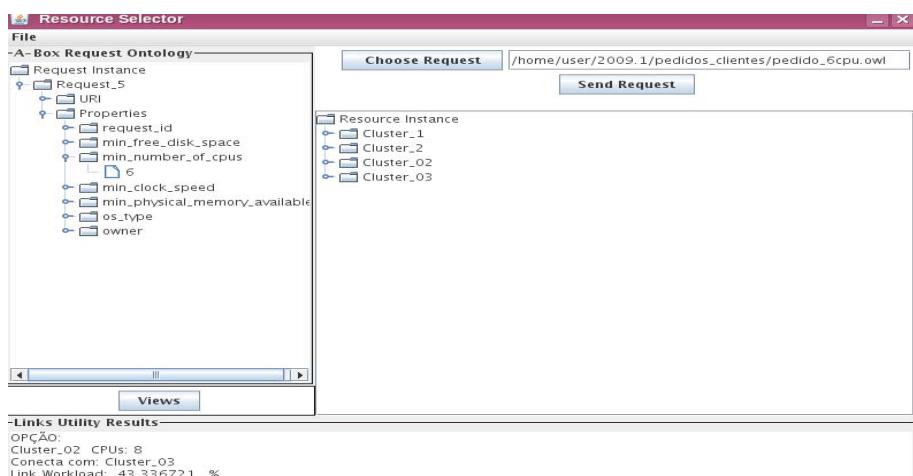


Fig. 6. Quering 6 processors. New system with dynamic information available.

The next experiment represents the previous one considering all new features of the new proposal. A comprehensive picture of available multi-cluster is presented in Fig. 6. This figure shows the same query, but now executed under the new proposal.

It is clear from Fig. 6 that multi-cluster exist to support the query. The query under the new proposal considers all processors that exist inside all clusters that have one communication link. The final result also considers the best communication path, thus suggesting the job splitting. The Result panel, on the left side down in Fig. 6, shows all clusters that satisfy the query. Also, the LinksResults highlights the best calculated solution.

5 Conclusions and Future Work

In this paper we have presented a novel dynamic resource matching approach, which considers computer resources and communication links characteristics, to gather nodes from multi-cluster configurations as a grid environment. The idea is to provide a standard interface to create a grid environment to execute distributed and parallel applications. The proposal can be understood as an advance resource (AR) function to execute applications in multi-cluster configurations.

The work employed ontology-based and fuzzy logic paradigms. The ontology-based approach was utilized for a standard description of resources to characterize a multi-cluster environment. Also, fuzzy logic algorithms were used to monitor resources and also to dynamically match end-users's requests and available computer resources and communication link workloads.

The proposal was tested using the SimGrid simulator. The proposal has reached a successful level, providing interesting multi-cluster configuration options to application programmers.

As a future work we are planning to implement co-reservation and co-allocation algorithms and build an entire meta-scheduler environment. This meta-scheduler would consider different types of multi-cluster configurations, including Ad Hoc nodes. We are also planning a fault-tolerance module to provide a checkpoint facility for more complex applications. Finally, we are also planning a component to automatically execute an application in the best selected configuration.

References

1. Manager/Moab Workload Manager,
<http://www.clusterresources.com/pages/products.php>
2. PBS, <http://www.pbsgridworks.com/>
3. Platform Computing Inc.: LSF Product Suite,
<http://www.platform.com/Products>
4. IBM Corp.: LoadLeveler, <http://www-03.ibm.com/systems/clusters/software/>
5. Dalheimer, M., Pfreundt, F.-J., Merz, P.: Calana: a general-purpose agent-based grid scheduler. In: 14th IEEE International Symposium on High Performance Distributed Computing, HPDC-14, pp. 279–280 (2005)

6. Agarwal, V., Dasgupta, G., Dasgupta, K., Purohit, A., Viswanathan, B.: DECO: Data Replication and Execution CO-scheduling for Utility Grids. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 52–65. Springer, Heidelberg (2006)
7. GridWay Metascheduler, <http://www.gridway.org>
8. Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y., Sekiguchi, S.: GridARS: An advance reservation-based grid co-allocation framework for distributed computing and network resources. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2007. LNCS, vol. 4942, pp. 152–168. Springer, Heidelberg (2008)
9. Eickermann, T., Frings, W., Wieder, P., Waldrich, O., Ziegler, W.: Co-allocation of MPI Jobs with the VIOLA Grid MetaScheduling Framework, CoreGRID. Technical Report Number TR-0081, May 28 (2007)
10. CSF, http://www.globus.org/grid_software/computation/csf.php
11. Changtao, Q.: A Grid Advance Reservation Framework for Co-allocation and Co-reservation Across Heterogeneous Local Resource Management Systems. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 770–779. Springer, Heidelberg (2008)
12. Zhou, L., Chen, H., Mao, Y.: A semantic-based framework for virtual organization management. In: The Third ChinaGrid Annual Conference, ChinaGrid 2008, August 2008, pp. 294–299 (2008)
13. Ejarque, J., de Palol, M., Goiri, I., Julia, F., Guitart, J., Torres, J., Badia, R.: Using semantics for resource allocation in computing service providers. In: Conference on Services Computing, SCC 2008, July 2008, pp. 583–587 (2008)
14. Silva, A.P.C., Dantas, M.A.R.: A Selector of Grid Resources based on the Semantic Integration of Multiple Ontologies. In: Proc. of the 19th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2007, Gramado, Brazil, pp. 143–150 (2007)
15. Qin, J., Bauer, M.A.: An Improved Job Co-Allocation Strategy in Multiple HPC Clusters. In: 21st International Symposium on High Performance Computing Systems and Applications (HPCS 2007), p. 18 (2007)
16. Qin, J.: Job Co-Allocation Strategies in Multiple HPC Clusters, PhD Thesis, University of Western Ontario, Department of Computer Science
17. Jena, <http://jena.sourceforge.net/inference/>
18. Simgrid, <http://simgrid.gforge.inria.fr/>

Domain Decomposition of Stochastic PDEs: A Novel Preconditioner and Its Parallel Performance

Waad Subber* and Abhijit Sarkar**

Carleton University, Ottawa, Ontario K1S5B6, Canada

Abstract. A parallel iterative algorithm is described for efficient solution of the Schur complement (interface) problem arising in the domain decomposition of stochastic partial differential equations (SPDEs) recently introduced in [12]. The iterative solver avoids the explicit construction of both local and global Schur complement matrices. An analog of Neumann-Neumann domain decomposition preconditioner is introduced for SPDEs. For efficient memory usage and minimum floating point operation, the numerical implementation of the algorithm exploits the multi-level sparsity structure of the coefficient matrix of the stochastic system. The algorithm is implemented using PETSc parallel libraries. Parallel graph partitioning tool ParMETIS is used for optimal decomposition of the finite element mesh for load balancing and minimum interprocessor communication. For numerical demonstration, a two dimensional elliptic SPDE with non-Gaussian random coefficients is tackled. The strong and weak scalability of the algorithm is investigated using Linux cluster.

Keywords: Domain Decomposition Method, Schur Complement System, Neumann-Neumann Preconditioner, Stochastic PDEs, Polynomial Chaos Expansion.

1 Introduction

Domain decomposition method of stochastic PDEs has recently been proposed [12] to quantify uncertainty in large scale systems. The methodology is based on Schur complement based geometric decomposition and an orthogonal decomposition and projection of the stochastic processes. The resulting *extended* Schur complement system is a coupled system with block sparsity structure. In contrast to a deterministic PDE, the size of Schur complement (interface) problem of SPDE grows rapidly as the number of subdomains and the order of polynomial chaos (PC) expansion is increased. The solution of the interface problem primarily dictates the parallel performance of the domain decomposition algorithm. In this paper, we propose an iterative parallel solver equipped with a local

* Graduate Student, Department of Civil and Environmental Engineering, Carleton University.

** Associate Professor and Canada Research Chair, Department of Civil and Environmental Engineering, Carleton University.

preconditioner to tackle the extended Schur complement problem for SPDEs. More specifically, we adopt a parallel preconditioned conjugate gradient method (PCGM) to solve the interface problem without explicitly constructing the schur complement system. A Neumann-Neumann preconditioner [34] is introduced for SPDEs to enhance the performance of the PCGM solver. The implementation of the method requires a local solve of a stochastic Dirichlet problem followed by a local solve of a stochastic Neumann problem in each iteration of the PCGM solver. The multilevel sparsity structure of the coefficient matrix of the stochastic system, namely (a) the sparsity structure due to the finite element discretization and (b) the block sparsity structure due to the orthogonal representation and projection of the stochastic processes is exploited in the implementation of the algorithm. PETSc [5] parallel libraries is used for efficient distributed implementation of the algorithm. A parallel graph partitioning tool, namely ParMATIS [6] is used for optimal decomposition of the geometric domain for load balancing and minimum interprocessor communication. The parallel performance of the algorithm is studied for a two dimensional stochastic elliptic PDE with non-Gaussian random coefficients. In particular, the strong and the weak scalability of the algorithm is investigated using a Linux cluster.

2 Uncertainty Representation by Stochastic Processes

We assume the data induces a representation of the model parameters as random variables and processes which span the Hilbert space \mathcal{H}_G . A set of basis functions $\{\xi_i\}$ is identified to characterize this space using Karhunen-Loeve expansion. The state of the system resides in the Hilbert space \mathcal{H}_L with basis functions $\{\Psi_i\}$ being identified with the polynomial chaos expansion (PC). The Karhunen-Loeve expansion of a stochastic process $\alpha(x, \theta)$ is based on the spectral expansion of its covariance function $R_{\alpha\alpha}(x, y)$. The expansion takes the following form

$$\alpha(x, \theta) = \bar{\alpha}(x) + \sum_{i=1}^{\infty} \sqrt{\lambda_i} \xi_i(\theta) \phi_i(x), \quad (1)$$

where $\bar{\alpha}(x)$ is the mean of the stochastic process, θ represents the random dimension, and $\xi_i(\theta)$ is a set of uncorrelated (but not generally independent for non-Gaussian processes) random variables, $\phi_i(x)$ are the eigenfunctions and λ_i are the eigenvalues of the covariance kernel which can be obtained as the solution to the following integral equation

$$\int_{\infty} R_{\alpha\alpha}(x, y) \phi_i(y) dy = \lambda_i \phi_i(x). \quad (2)$$

The covariance function of the solution process is not known a priori, and hence the Karhunen-Loeve expansion cannot be used to represent it. Therefore, a generic basis, that is complete in the space of all second-order random variables will be identified and used in the approximation process. Because the solution process is a function of the material properties, nodal solution variables, denoted

by $u(\theta)$ can be formally expressed as some nonlinear functional of the set $\xi_i(\theta)$ used to represent the material stochasticity. It has been shown that this functional dependence can be expanded in terms of polynomials in Gaussian random variables, referred to as polynomial chaos[8]. Namely

$$\mathbf{u}(\theta) = \sum_{j=0}^N \Psi_j(\theta) \mathbf{u}_j. \quad (3)$$

These polynomials are orthogonal in the sense that their inner product $\langle \Psi_j \Psi_k \rangle$, which is defined as the statistical average of their product, is equal to zero for $j \neq k$.

3 Stochastic Domain Decomposition

Consider an elliptic stochastic PDE defined on a domain Ω with a given boundary conditions on $\partial\Omega$. The finite element approximation of the PDE leads to the following discrete linear system

$$\mathbf{A}(\theta) \mathbf{u}(\theta) = \mathbf{f}. \quad (4)$$

For large scale systems, Eq.(4) can be solved efficiently using domain decomposition method [12].

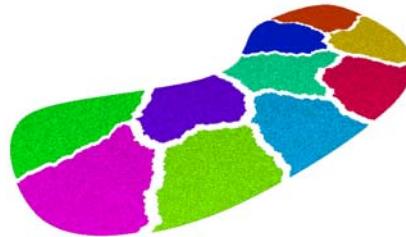


Fig. 1. A typical domain partitioned into n non-overlapping subdomains

Assume that the spatial domain Ω , as shown schematically in Fig.(1), is partitioned into n_s non-overlapping subdomains $\{\Omega_s, 1 \leq s \leq n_s\}$ such that

$$\Omega = \bigcup_{s=1}^{n_s} \Omega_s, \quad \Omega_s \cap \Omega_r = \emptyset \quad s \neq r \quad (5)$$

and

$$\Gamma = \bigcup_{s=1} \Gamma_s \text{ where } \Gamma_s = \partial\Omega_s \setminus \partial\Omega \quad (6)$$

For a typical subdomain, the solution vector $\mathbf{u}^s(\theta)$ can be split into two sub-vectors $\mathbf{u}_I^s(\theta)$ and $\mathbf{u}_\Gamma^s(\theta)$ which constitute the interior and interface unknowns

respectively. Consequently, the equilibrium equation for the subsystem can be represented as

$$\begin{bmatrix} \mathbf{A}_{II}^s(\theta) & \mathbf{A}_{I\Gamma}^s(\theta) \\ \mathbf{A}_{\Gamma I}^s(\theta) & \mathbf{A}_{\Gamma\Gamma}^s(\theta) \end{bmatrix} \begin{Bmatrix} \mathbf{u}_I^s(\theta) \\ \mathbf{u}_\Gamma^s(\theta) \end{Bmatrix} = \begin{Bmatrix} \mathbf{f}_I^s \\ \mathbf{f}_\Gamma^s \end{Bmatrix}. \quad (7)$$

The polynomial chaos expansion can be used to represent the uncertainty in the model parameters. Consequently, the stiffness matrix is approximated as

$$\sum_{i=0}^L \Psi_i \begin{bmatrix} \mathbf{A}_{II,i}^s & \mathbf{A}_{I\Gamma,i}^s \\ \mathbf{A}_{\Gamma I,i}^s & \mathbf{A}_{\Gamma\Gamma,i}^s \end{bmatrix} \begin{Bmatrix} \mathbf{u}_I^s(\theta) \\ \mathbf{u}_\Gamma^s(\theta) \end{Bmatrix} = \begin{Bmatrix} \mathbf{f}_I^s \\ \mathbf{f}_\Gamma^s \end{Bmatrix}, \quad (8)$$

where L represents the number of terms retained in the Karhunen-Loeve expansion in Eq.(II).

A rectangular matrix R_s of size $(n_\Gamma^s \times n_\Gamma)$ defines a restriction operator that maps the global interface unknown u_Γ to the local interface vector u_I^s as

$$u_I^s = R_s u_\Gamma. \quad (9)$$

In parallel implementation R_s acts as a *scatter* operator while R_s^T acts as a *gather* operator and is not constructed explicitly.

Enforcing the transmission conditions (compatibility and equilibrium) along the interfaces, the global equilibrium equation of the stochastic system can be expressed in the following block linear systems of equations:

$$\sum_{i=0}^L \Psi_i \begin{bmatrix} \mathbf{A}_{II,i}^1 & \dots & 0 & \mathbf{A}_{I\Gamma,i}^1 \mathbf{R}_1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & \mathbf{A}_{II,i}^{n_s} & \mathbf{A}_{I\Gamma,i}^{n_s} \mathbf{R}_{n_s} \\ \mathbf{R}_1^T \mathbf{A}_{\Gamma I,i}^1 \dots \mathbf{R}_{n_s}^T \mathbf{A}_{\Gamma I,i}^{n_s} \sum_{s=1}^{n_s} \mathbf{R}_s^T \mathbf{A}_{I\Gamma,i}^s \mathbf{R}_s \end{bmatrix} \begin{Bmatrix} \mathbf{u}_I^1(\theta) \\ \vdots \\ \mathbf{u}_I^{n_s}(\theta) \\ \mathbf{u}_\Gamma(\theta) \end{Bmatrix} = \begin{Bmatrix} \mathbf{f}_I^1 \\ \vdots \\ \mathbf{f}_I^{n_s} \\ \sum_{s=1}^{n_s} \mathbf{R}_s^T \mathbf{f}_\Gamma^s \end{Bmatrix}. \quad (10)$$

The solution process can be expanded using the same polynomial chaos basis as

$$\begin{Bmatrix} \mathbf{u}_I^1(\theta) \\ \vdots \\ \mathbf{u}_I^{n_s}(\theta) \\ \mathbf{u}_\Gamma(\theta) \end{Bmatrix} = \sum_{j=0}^N \Psi_j(\theta) \begin{Bmatrix} \hat{\mathbf{u}}_{I,j}^1 \\ \vdots \\ \hat{\mathbf{u}}_{I,j}^{n_s} \\ \hat{\mathbf{u}}_{\Gamma,j} \end{Bmatrix}. \quad (11)$$

Substituting Eq.(II) into Eq.(10) and performing the Galerkin projection to minimize the error over the space spanned by the chaos basis, the following coupled deterministic systems of equations is obtained

$$\begin{bmatrix} \mathcal{A}_{II}^1 & \dots & 0 & \mathcal{A}_{I\Gamma}^1 \mathcal{R}_1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & \mathcal{A}_{II}^{n_s} & \mathcal{A}_{I\Gamma}^{n_s} \mathcal{R}_{n_s} \\ \mathcal{R}_1^T \mathcal{A}_{\Gamma I}^1 \dots \mathcal{R}_{n_s}^T \mathcal{A}_{\Gamma I}^{n_s} \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{A}_{I\Gamma}^s \mathcal{R}_s \end{bmatrix} \begin{Bmatrix} \hat{\mathcal{U}}_I^1 \\ \vdots \\ \hat{\mathcal{U}}_I^{n_s} \\ \hat{\mathcal{U}}_\Gamma \end{Bmatrix} = \begin{Bmatrix} \mathcal{F}_I^1 \\ \vdots \\ \mathcal{F}_I^{n_s} \\ \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{F}_\Gamma^s \end{Bmatrix}, \quad (12)$$

where we define the following variables

$$\begin{aligned} [\mathcal{A}_{\alpha\beta}^s]_{jk} &= \sum_{i=0}^L \langle \Psi_i \Psi_j \Psi_k \rangle \mathbf{A}_{\alpha\beta,i}^s , \quad \mathcal{F}_{\alpha,k}^s = \langle \Psi_k \mathbf{f}_\alpha^s \rangle, \\ \hat{\mathcal{U}}_I^m &= (\hat{\mathbf{u}}_{I,0}^m, \dots, \hat{\mathbf{u}}_{I,N}^m)^T, \quad \hat{\mathcal{U}}_\Gamma = (\hat{\mathbf{u}}_{\Gamma,0}, \dots, \hat{\mathbf{u}}_{\Gamma,N})^T, \end{aligned} \quad (13)$$

where the subscripts α and β represent the index I and Γ . The coefficient matrix in Eq. (12) is of order $n(N+1) \times n(N+1)$ where n and $(N+1)$ denote the number of the physical degrees of freedom for a typical subdomain and chaos coefficients, respectively. The stochastic counterpart of the restriction operator in Eq. (12) takes the following form

$$\mathcal{R}_s = \text{blockdiag}(R_s^0, \dots, R_s^N), \quad (14)$$

where (R_s^0, \dots, R_s^N) are the deterministic restriction operators given by Eq. (9).

A block Gaussian elimination reduces the system in Eq. (12) to the following Schur complement system for the interface variable $\hat{\mathcal{U}}_\Gamma$ as

$$\mathcal{S} \hat{\mathcal{U}}_\Gamma = \mathcal{G}_\Gamma, \quad (15)$$

where the global Schur complement matrix \mathcal{S} is given by

$$\mathcal{S} = \sum_{s=1}^{n_s} \mathcal{R}_s^T [\mathcal{A}_{\Gamma\Gamma}^s - \mathcal{A}_{\Gamma I}^s (\mathcal{A}_{II}^s)^{-1} \mathcal{A}_{I\Gamma}^s] \mathcal{R}_s, \quad (16)$$

and the corresponding right hand side vector \mathcal{G}_Γ is

$$\mathcal{G}_\Gamma = \sum_{s=1}^{n_s} \mathcal{R}_s^T [\mathcal{F}_\Gamma^s - \mathcal{A}_{\Gamma I}^s (\mathcal{A}_{II}^s)^{-1} \mathcal{F}_I^s]. \quad (17)$$

Once the interface unknowns $\hat{\mathcal{U}}_\Gamma$ is available, the interior unknowns can be obtained concurrently by solving the interior problem on each subdomain as

$$\mathcal{A}_{II}^s \hat{\mathcal{U}}_I^s = \mathcal{F}_I^s - \mathcal{A}_{\Gamma I}^s \mathcal{R}_s \hat{\mathcal{U}}_\Gamma. \quad (18)$$

The primary task at hand is to tackle the Schur complement system in Eq. (15) using an iterative solver that demonstrates scalability in parallel computers.

In practice, it is expensive to construct Schur complement matrix \mathcal{S} explicitly in Eq. (16). In this paper, a parallel preconditioned conjugate gradient method is used to solve the Schur complement system which avoids the explicit construction of the Schur complement matrix [3, 17].

4 The Extended Schur Complement Matrix for SPDEs

In this section, some special features of the extended Schur complement matrix for SPDEs are highlighted.

The extended Schur complement matrix for SPDEs has two distinct level of sparsity structure. The first level relates to the finite element discretization whereas the second level arises due to the stochastic features of the problem. In the computer implementation of the algorithm, this multilevel sparsity structure is exploited in order to minimize the storage requirements and to reduce the unnecessary floating point operations associated with zero-valued elements.

Fig.(2) shows the non-zero block pattern of the Schur complement matrix for the first, second and third order PC expansion. The size of Schur complement matrix is $[n_\Gamma(N+1)] \times [n_\Gamma(N+1)]$ where n_Γ is the number of the global interface unknowns and $(N+1)$ is the order of PC expansion. The size of each individual block is $n_\Gamma \times n_\Gamma$ and the number of blocks is $(N+1)$.

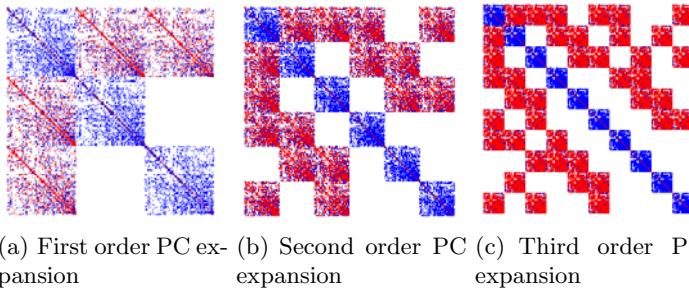


Fig. 2. The sparsity structure of the extended Schur complement matrix for SPDEs

Note that we do not construct or store the extended Schur complement matrix. Instead we only store the non-zero elements of the subdomain stiffness matrices $\mathcal{A}_{\alpha\beta}^s$ in Eq.(16). PETSc[5] sparse matrix (compressed row) format is used for efficient memory management. For better performance of PETSc matrix assembly process, the memory required for the sparse matrix storage need to be preallocated by setting the number of non-zeros entries per each row. This task can be accomplished by defining an integer array of size n_α where α represents the index I and Γ . The entries of this array are obtained by looping over the rows of the $\mathcal{A}_{\alpha\beta,i}^s$ matrix in Eq.(13) and counting the number of non-zeros for each row. This array represents the first sparsity level of the stochastic matrix. A second array of size $(N+1)$ containing the number of non-zeros block per row is constructed by counting the non-zeros coefficients of the $\langle \Psi_i \Psi_j \Psi_k \rangle$ (see Eq.(13)) which provides the second level of sparsity. Finally, an array of size $n_\alpha(N+1)$ is formed by multiplying the first array by the corresponding element of the second array. This array contains the sparsity pattern of the stochastic submatrices $\mathcal{A}_{\alpha\beta}^s$ in Eq.(16). Recall that $\mathcal{A}_{\alpha\beta}^s$ relates to \mathcal{S} through Eq.(16).

5 A Neumann-Neumann Preconditioner for SPDEs

In this section, an analog of Neumann-Neumann domain decomposition preconditioner[34] is introduced for SPDEs. The implementation of the preconditioner requires a local solve of a stochastic Dirichlet problem followed by a

local solve of a stochastic Neumann problem in each iteration of the conjugate gradient iterative solver. The procedure can be described as follow.

For a typical subdomain Ω_s , the equilibrium equation can be written as

$$\begin{bmatrix} \mathcal{A}_{II}^s & \mathcal{A}_{IG}^s \\ \mathcal{A}_{GI}^s & \mathcal{A}_{GG}^s \end{bmatrix} \begin{Bmatrix} \hat{\mathcal{U}}_I^s \\ \hat{\mathcal{U}}_G^s \end{Bmatrix} = \begin{Bmatrix} \mathcal{F}_I^s \\ \mathcal{F}_G^s \end{Bmatrix}. \quad (19)$$

Starting with an initial guess $\hat{\mathcal{U}}_{\Gamma_j}$ on the boundary interface (where j is the iteration number), a stochastic Dirichlet problem can be solved in parallel on each subdomain.

Hence, from the first row of Eq.(19), the interior degrees of freedom can be obtained as

$$\hat{\mathcal{U}}_I^s = [\mathcal{A}_{II}^s]^{-1} (\mathcal{F}_I^s - \mathcal{A}_{IG}^s \hat{\mathcal{U}}_G^s). \quad (20)$$

The corresponding local residual r_I^s can be computed from the second row of Eq.(19) as

$$r_I^s = \mathcal{A}_{GI}^s \hat{\mathcal{U}}_I^s + \mathcal{A}_{GG}^s \hat{\mathcal{U}}_G^s - \mathcal{F}_G^s. \quad (21)$$

Using Eq.(20) and Eq.(21), $\hat{\mathcal{U}}_I^s$ can be eliminated to obtain

$$r_I^s = (\mathcal{A}_{GG}^s - \mathcal{A}_{GI}^s [\mathcal{A}_{II}^s]^{-1} \mathcal{A}_{IG}^s) \hat{\mathcal{U}}_G^s - (\mathcal{F}_G^s - \mathcal{A}_{GI}^s [\mathcal{A}_{II}^s]^{-1} \mathcal{F}_I^s). \quad (22)$$

The local residual in Eq.(22) can be rewritten as

$$r_I^s = \mathcal{S}_s \hat{\mathcal{U}}_G^s - \mathcal{G}_G^s, \quad (23)$$

where the local Schur complement matrix \mathcal{S}_s for the stochastic system is given by

$$\mathcal{S}_s = \mathcal{A}_{GG}^s - \mathcal{A}_{GI}^s [\mathcal{A}_{II}^s]^{-1} \mathcal{A}_{IG}^s, \quad (24)$$

and the corresponding right hand side vector \mathcal{G}_G^s is

$$\mathcal{G}_G^s = \mathcal{F}_G^s - \mathcal{A}_{GI}^s [\mathcal{A}_{II}^s]^{-1} \mathcal{F}_I^s. \quad (25)$$

The global equilibrium is enforced next by aggregating the local residuals from all the subdomains as

$$r_{\Gamma_j} = \sum_{s=1}^{n_s} \mathcal{R}_s^T r_I^s, \quad (26)$$

where r_{Γ_j} is the global residual vector at the j^{th} iteration which represents the *unbalanced* force due to the erroneous initial guess.

Let \mathcal{D}_s represent a diagonal scaling matrix that derives from the partition of unity as

$$\sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{D}_s \mathcal{R}_s = \mathcal{I}. \quad (27)$$

The diagonal entries of \mathcal{D}_s are the reciprocal of the number of subdomains that share the boundary degrees of freedom.

The unbalanced force given by Eq.(26) can now be distributed over the sub-domains as

$$r_{\Gamma}^s = \mathcal{D}_s \mathcal{R}_s r_{\Gamma_j}. \quad (28)$$

The corresponding unknown, $\hat{\mathcal{U}}_{\Gamma}^s$ due to the unbalanced force given by Eq.(28) can be computed by solving a local Schur complement system as

$$\hat{\mathcal{U}}_{\Gamma}^s = [\mathcal{S}_s]^{-1} r_{\Gamma}^s. \quad (29)$$

The inversion of the Schur complement matrix in Eq.(29) is equivalent to solving the following local stochastic Neumann problem

$$\begin{bmatrix} \mathcal{A}_{II}^s & \mathcal{A}_{I\Gamma}^s \\ \mathcal{A}_{\Gamma I}^s & \mathcal{A}_{\Gamma\Gamma}^s \end{bmatrix} \begin{Bmatrix} \mathcal{X}^s \\ \hat{\mathcal{U}}_{\Gamma}^s \end{Bmatrix} = \begin{Bmatrix} 0 \\ r_{\Gamma}^s \end{Bmatrix}. \quad (30)$$

Using Eq.(28), Eq.(29) can be rewritten as

$$\hat{\mathcal{U}}_{\Gamma}^s = [\mathcal{S}_s]^{-1} \mathcal{D}_s \mathcal{R}_s r_{\Gamma_j}, \quad (31)$$

The new approximate solution $\hat{\mathcal{U}}_{\Gamma_{j+1}}$ can be obtained by updating the initial guess $\hat{\mathcal{U}}_{\Gamma_j}$ with the average local solutions $\hat{\mathcal{U}}_{\Gamma}^s$ as

$$\hat{\mathcal{U}}_{\Gamma_{j+1}} = \hat{\mathcal{U}}_{\Gamma_j} + \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{D}_s^T \hat{\mathcal{U}}_{\Gamma}^s. \quad (32)$$

Equivalently, Eq.(32) can be rewritten as

$$e_{\Gamma_j} = \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{D}_s^T [\mathcal{S}_s]^{-1} \mathcal{D}_s \mathcal{R}_s r_{\Gamma_j}, \quad (33)$$

where $e_{\Gamma_j} = \hat{\mathcal{U}}_{\Gamma_{j+1}} - \hat{\mathcal{U}}_{\Gamma_j}$ is the error at the j^{th} iteration.

Eq.(33) involving the error term e_{Γ_j} can be expressed as

$$r_{\Gamma_j} = \mathcal{M}^{-1} r_{\Gamma_j}, \quad (34)$$

where the Neumann-Neumann preconditioner \mathcal{M}^{-1} for the stochastic system is

$$\mathcal{M}^{-1} = \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{D}_s^T [\mathcal{S}_s]^{-1} \mathcal{D}_s \mathcal{R}_s. \quad (35)$$

6 Parallel Implementation

The global extended Schur complement matrix for SPDEs is symmetric and positive definite. Therefore the preconditioned conjugate gradient method (PCGM) can be applied to solve the global extended Schur complement system.

In PCGM, the extended Schur complement matrix need not be constructed explicitly as only its effect on a vector is required [37]. This matrix-vector product can be obtained concurrently by solving a local stochastic Dirichlet problem

on each subdomain and then gathering the resulting vectors. Similarly the effect of Neumann-Neumann preconditioner can be achieved by tackling a local stochastic Neumann problem on each subdomain in parallel and the resulting vectors are then aggregated. In what follows is a concise description of this procedure.

For the j^{th} iteration in the PCGM algorithm, the matrix-vector product involving $\mathcal{Q}_j = \mathcal{S}\mathcal{P}_j$ (see step 9 of algorithm: 6.1) can be computed by first scattering the global vector \mathcal{P}_j to the local \mathcal{P}^s (step 8 of algorithm: 6.1) as

$$\mathcal{P}^s = \mathcal{R}_s \mathcal{P}_j. \quad (36)$$

The action of the local Schur complement matrix \mathcal{S}_s on the local vector \mathcal{P}^s can be obtained by considering the following subdomain system

$$\begin{bmatrix} \mathcal{A}_{II}^s & \mathcal{A}_{I\Gamma}^s \\ \mathcal{A}_{\Gamma I}^s & \mathcal{A}_{\Gamma\Gamma}^s \end{bmatrix} \begin{Bmatrix} \mathcal{X}^s \\ \mathcal{P}^s \end{Bmatrix} = \begin{Bmatrix} 0 \\ \mathcal{Q}^s \end{Bmatrix}. \quad (37)$$

From the first row of Eq.(37), a stochastic Dirichlet problem can be solved as

$$\mathcal{X}^s = -[\mathcal{A}_{II}^s]^{-1} \mathcal{A}_{I\Gamma}^s \mathcal{P}^s. \quad (38)$$

Substituting the value of \mathcal{X}^s in the second row of Eq.(37), the following expression for \mathcal{Q}^s is obtained

$$\mathcal{Q}^s = [\mathcal{A}_{\Gamma\Gamma}^s - \mathcal{A}_{\Gamma I}^s (\mathcal{A}_{II}^s)^{-1} \mathcal{A}_{I\Gamma}^s] \mathcal{P}^s, \quad (39)$$

which is equivalent to

$$\mathcal{Q}^s = \mathcal{S}_s \mathcal{P}^s. \quad (40)$$

The action of the global extended stochastic Schur complement matrix on the global vector \mathcal{P}_j can be obtained by gathering the local vectors \mathcal{Q}^s as

$$\mathcal{Q}_j = \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{Q}^s. \quad (41)$$

Using Eq.(36) and Eq.(40), \mathcal{Q}_j in Eq.(41) can be expressed as

$$\mathcal{Q}_j = \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{S}_s \mathcal{P}^s = \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{S}_s \mathcal{R}_s \mathcal{P}_j. \quad (42)$$

Next, the preconditioned residual vector \mathcal{Z}_{j+1} (see step 15 of algorithm: 6.1) can be computed by solving a local stochastic Neumann problem. To tackle this Neumann problem, the global residual vector $r_{\Gamma_{j+1}}$ is distributed over the subdomains (step 14 of algorithm: 6.1) as

$$r_{\Gamma}^s = \mathcal{D}_s \mathcal{R}_s r_{\Gamma_{j+1}}. \quad (43)$$

Next, a local stochastic Neumann problem is tackled on each subdomain to find the local preconditioned vector \mathcal{Z}^s as

$$\mathcal{Z}^s = [\mathcal{S}_s]^{-1} r_{\Gamma}^s. \quad (44)$$

In terms of the global residual $r_{\Gamma_{j+1}}$, \mathcal{Z}^s can be expressed as

$$\mathcal{Z}^s = [\mathcal{S}_s]^{-1} \mathcal{D}_s \mathcal{R}_s r_{\Gamma_{j+1}}. \quad (45)$$

The global preconditioned residual vector, \mathcal{Z}_{j+1} can be calculated by aggregating the local vectors \mathcal{Z}^s as

$$\mathcal{Z}_{j+1} = \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{D}_s^T \mathcal{Z}^s, \quad (46)$$

Alternatively, \mathcal{Z}_{j+1} can be expressed as

$$\mathcal{Z}_{j+1} = \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{D}_s^T [\mathcal{S}_s]^{-1} \mathcal{D}_s \mathcal{R}_s r_{\Gamma_{j+1}}. \quad (47)$$

Clearly, the Neumann-Neumann preconditioner in Eq.(47) is identified to be

$$\mathcal{M}^{-1} = \sum_{s=1}^{n_s} \mathcal{M}_s^{-1}, \quad (48)$$

where

$$\mathcal{M}_s^{-1} = \mathcal{R}_s^T \mathcal{D}_s^T [\mathcal{S}_s]^{-1} \mathcal{D}_s \mathcal{R}_s. \quad (49)$$

The algorithm of the parallel PCGM applied to solve the extended Schur complement system in Eq.(15) is outlined below:

Algorithm. 6.1 Parallel PCGM solver to Schur complement system

1. $\hat{\mathcal{U}}_{\Gamma_0} := 0$
2. *Gather* $r_{\Gamma_0} := \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{G}_s^s$
3. *Scatter* r_{Γ_0}
4. *Gather* $\mathcal{Z}_0 = \mathcal{M}^{-1} r_{\Gamma_0} := \sum_{s=1}^{n_s} \mathcal{M}_s^{-1} r_{\Gamma_0}$
5. $\mathcal{P}_0 := \mathcal{Z}_0$
6. $\rho_0 := (r_{\Gamma_0}, \mathcal{Z}_0)$
7. *For* $j = 0, 1, \dots$, until convergence *Do*
8. *Scatter* \mathcal{P}_j
9. *Gather* $\mathcal{Q}_j = \mathcal{S} \mathcal{P}_j := \sum_{s=1}^{n_s} \mathcal{R}_s^T \mathcal{S}_s \mathcal{R}_s \mathcal{P}_j$
10. $\rho_{tmp} := (\mathcal{P}_j, \mathcal{Q}_j)$
11. $\alpha := \rho_j / \rho_{tmp}$
12. $\hat{\mathcal{U}}_{\Gamma_{j+1}} := \hat{\mathcal{U}}_{\Gamma_j} + \alpha \mathcal{P}_j$
13. $r_{\Gamma_{j+1}} := r_{\Gamma_j} - \alpha \mathcal{Q}_j$
14. *Scatter* $r_{\Gamma_{j+1}}$
15. *Gather* $\mathcal{Z}_{j+1} = \mathcal{M}^{-1} r_{\Gamma_{j+1}} := \sum_{s=1}^{n_s} \mathcal{M}_s^{-1} r_{\Gamma_{j+1}}$
16. $\rho_{j+1} := (r_{\Gamma_{j+1}}, \mathcal{Z}_{j+1})$
17. $\beta_j := \rho_{j+1} / \rho_j$
18. $\mathcal{P}_{j+1} = \mathcal{P}_j + \beta_j \mathcal{Z}_{j+1}$
19. *EndDo*

7 Numerical Results

In this section, we analyze a stationary (time-independent) stochastic PDE as a numerical illustration to the aforementioned mathematical framework. In particular, we consider a two dimensional Poisson equation with randomly heterogeneous permeability coefficients,

$$\frac{\partial}{\partial x} [c_x(x, y, \theta) \frac{\partial u(x, y, \theta)}{\partial x}] + \frac{\partial}{\partial y} [c_y(x, y, \theta) \frac{\partial u(x, y, \theta)}{\partial y}] = f(x, y) \quad \text{in } \Omega, \quad (50)$$

where the forcing term is

$$f(x, y) = 150\pi^2 \sin(\pi x) \sin(2\pi y). \quad (51)$$

For simplicity, a homogeneous Dirichlet boundary condition is imposed as

$$u(x, y, \theta) = 0 \quad \text{on } \partial\Omega. \quad (52)$$

Uncertainty stems from the random permeability coefficients $c_x(x, y, \theta)$ and $c_y(x, y, \theta)$ which are modeled as independent lognormal random variables with mean 2.8 and standard deviation 0.712. The coefficient of variation (*CoV*) equal to 0.25. The theoretical framework is however very general and can handle the spatial variability of the permeability coefficients as stochastic processes. The simplified representation of uncertainty as random variables (in contrast to stochastic processes) is adequate to demonstrate the usefulness of the theoretical formulation without undue programming complexity. Finite element discretization with linear triangular elements results in 250,451 elements and 126,723 nodes. The random permeability coefficients and the response are represented by third order polynomial chaos expansion ($L = 7$, $N = 9$) leading to a linear system of order 1,267,230.



Fig. 3. The FEM mesh

In PCGM implementation, the forcing term is taken to be the initial residual. The iterations are terminated when the ratio of L_2 norms of the current and the initial residual is less than 10^{-5}

$$\frac{\|\mathcal{G}_\Gamma^k - \mathcal{S}\mathcal{U}_\Gamma^k\|_2}{\|\mathcal{G}_\Gamma^0\|_2} \leqslant 10^{-5}. \quad (53)$$



Fig. 4. Mesh decomposition using ParMETIS

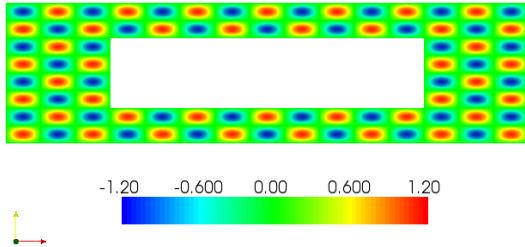


Fig. 5. The mean

Numerical experiments are performed in a Linux cluster with InfiniBand inter-connect (2 Quad-Core 3.0 GHz Intel Xeon processors and 32 GB of memory per node) using Message Passing Interface (MPI) [10]. The graph partitioning software ParMETIS [6] is used to decompose the finite element mesh and PETSc [5] parallel libraries handles the distributed implementation of matrix-vector operations.

Fig.(3) shows the finite element mesh while Fig.(4) presents a typical mesh decomposition of the geometric domain using ParMETIS graph partitioner. ParMETIS performs mesh decomposition using the criterion of load balancing among the CPUs while minimizing the size of interface boundary among subdomains.

The mean and the associated standard deviation of the solution process $u(x, y, \theta)$ are shown in Fig.(5) and Fig.(6) respectively. The maximum value of the coefficient of variation of the solution field is around 0.20 which highlights the effect of uncertainty. Note that the solution process $u(x, y, \theta)$ is non-Gaussian. Details of the stochastic features of the solution process, are shown in Fig.(7).

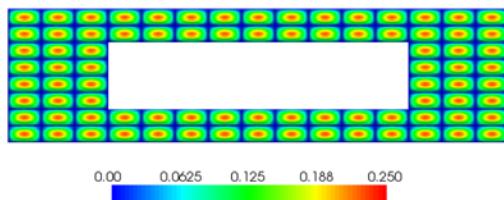


Fig. 6. The standard deviation

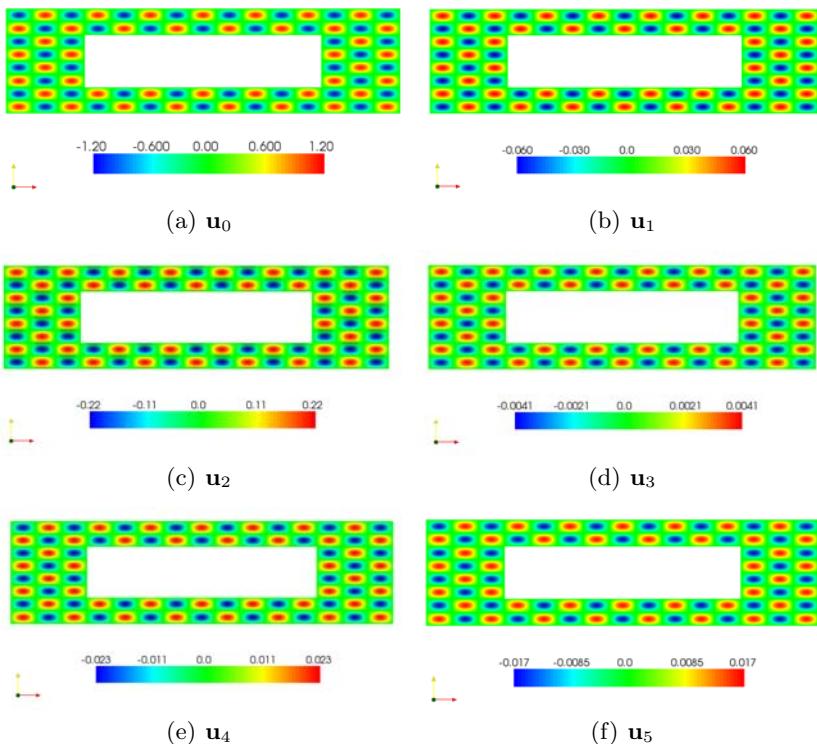


Fig. 7. Chaos coefficients: $\mathbf{u}_j, j = 0, \dots, 5$

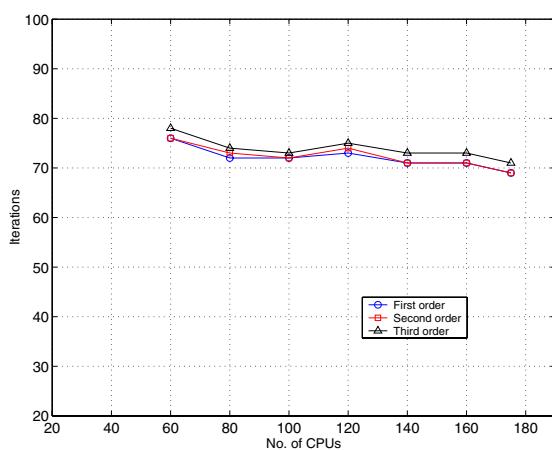


Fig. 8. Iteration count for fixed problem size

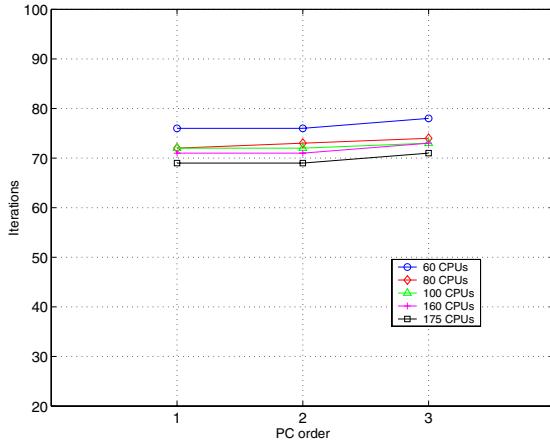


Fig. 9. Iteration count for first, second and third PC expansion order (fixed problem size)

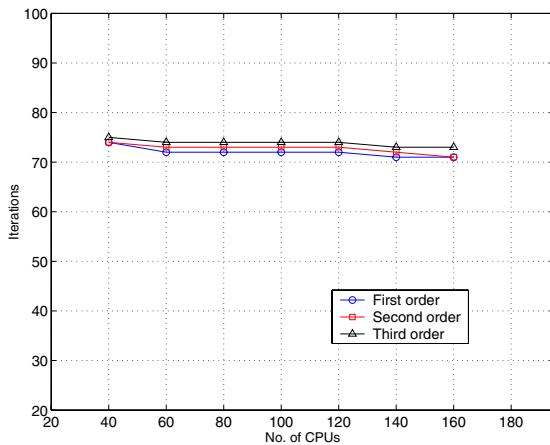


Fig. 10. Iteration count for fixed problem size per CPU

The chaos coefficients $\mathbf{u}_j(x, y)$ of the solution process $u(x, y, \theta)$ are presented in Fig. (7(a-f)). Note that \mathbf{u}_j for $j \geq 2$ represent the non-Gaussian contributions to the solution field $u(x, y, \theta)$.

Next, we investigate the strong and the weak scalability [11] of the algorithm. In Fig.(8), we fix the problem size in the spatial domain and increase the number of subdomains or CPUs/cores. The results obtained for first, second and third order PC expansion of the solution field are also identified in Fig.(8). Evidently the PCGM iteration count is roughly constant over the range of the subdomains or CPUs. Fig.(9) demonstrates the computational cost in terms of PCGM iterations incurred when the order of PC expansion is increased in order to capture

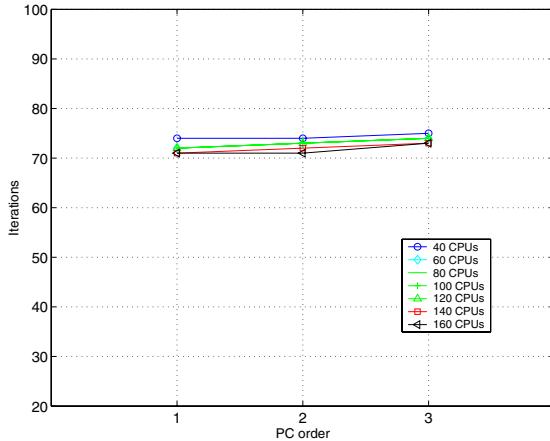


Fig. 11. Iteration count for first, second and third PC expansion order (fixed problem size per CPU)

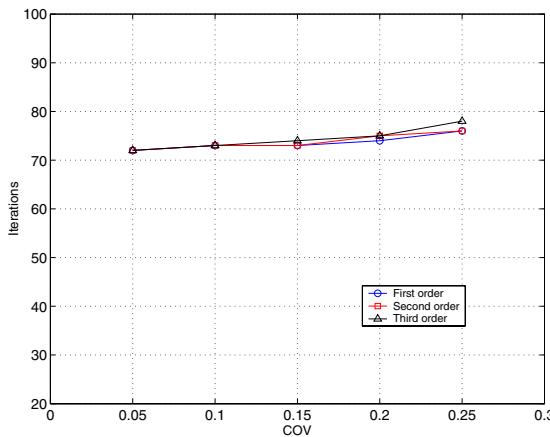


Fig. 12. Iteration count for the case of 60 CPUs and fixed problem size

the higher order nonlinear and non-Gaussian effects on the solution field due to parameter uncertainty. For a fixed problem size, a range of CPUs (60 to 175) is used to solve the problem. Fig. (9) indicates that the PCGM iteration counts remain fairly constant as the order of PC expansion increases. Furthermore, the number of PCGM iteration decreases as the number of CPUs are increased to solve the problem. These facts indicate the scalability of the algorithm. For the weak scalability study, we fix the subdomain size per CPU and increase the overall size of the problem by adding more CPUs. Fig. (10) shows the iteration count for a range of CPUs (40 to 160) for first, second and third order PC expansion. Clearly, Fig. (10) demonstrates that the iteration count is almost constant over

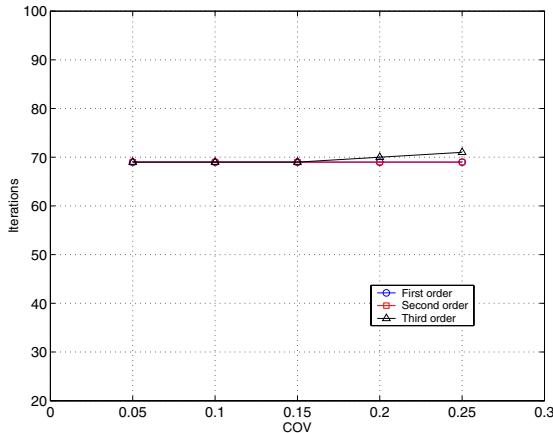


Fig. 13. Iteration count for the case 175 CPUs and fixed problem size

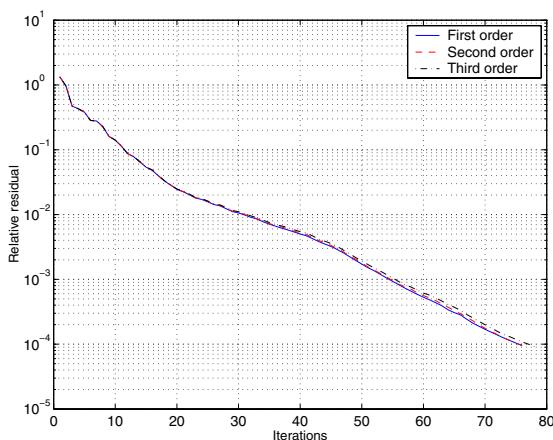


Fig. 14. The relative PCGM residual as a function of the iteration number for the case of 60 CPUs

the range of CPUs considered. Fig. (II) shows the PCGM iteration counts for fixed problem size per CPU when the order of PC expansion is increased. The number of PCGM iterations remain roughly constant indicating the algorithm is scalable.

Next we consider the performance of the algorithm when the strength of randomness of the system parameters is increased. Figs. (I2) & (I3) show the required number of PCGM iteration for 60 and 175 CPUs respectively when the CoV is increased for the random permeability parameters. Clearly, the strength of the randomness does not deteriorate the performance of the algorithm as the number of PCGM iteration is nearly constant.

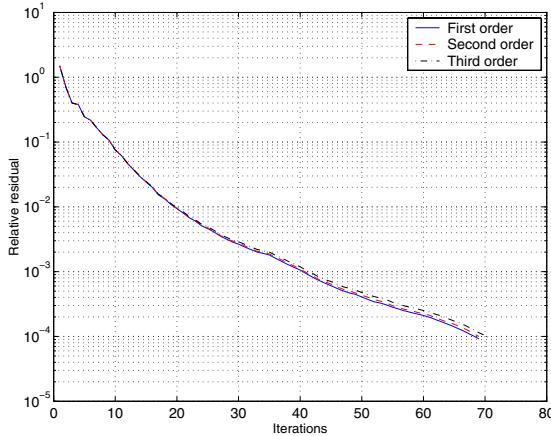


Fig. 15. The relative PCGM residual as a function of the iteration number for the case of 175 CPUs

For a fixed problem size, the convergence behavior of the algorithm is shown in Figs(14 & 15) for 60 and 175 CPUs for first, second and third PC expansion. The relative residual in PCGM iterations decays exponentially as the number of iteration increases highlighting the efficiency of the algorithm.

8 Conclusion

A parallel preconditioned conjugate gradient method (PCGM) is proposed to solve the extended Schur complement system arising in the domain decomposition of stochastic partial differential equations (SPDEs). The iterative solver is equipped with a stochastic version of Neumann-Neumann preconditioner. The algorithm achieves a convergence rate that is nearly independent of the coefficient of variation (*i.e.* the level of uncertainty) and the order of PC expansion. For the number of CPUs considered the algorithm demonstrates good (strong and weak) scalability. For large number of CPUs, the parallel performance of the algorithm may be enhanced by drawing a stochastic analog of multilevel (coarse- and fine-scale) preconditioners as used in the dual-primal domain decomposition [12][13] for deterministic PDEs. This aspect is currently being investigated by the authors.

Acknowledgments

The authors gratefully acknowledge the financial support from the Natural Sciences and Engineering Research Council of Canada through a Discovery Grant, Canada Research Chair Program, Canada Foundation for Innovation and Ontario Innovation Trust. Dr. Ali Rebaine for his help with ParMETIS graph-partitioning software.

References

1. Sarkar, A., Benabbou, N., Ghanem, R.: Domain Decomposition Of Stochastic PDEs: Theoretical Formulations. *International Journal for Numerical Methods in Engineering* 77, 689–701 (2009)
2. Sarkar, A., Benabbou, N., Ghanem, R.: Domain Decomposition Of Stochastic PDEs: Performance Study. *International Journal of High Performance Computing Applications* (to appear, 2009)
3. Toselli, A., Widlund, O.: Domain Decomposition Methods - Algorithms and Theory. Springer Series in Computational Mathematics, New York (2004)
4. Smith, B., Bjorstad, P., Gropp, W.: Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, Philadelphia (1996)
5. Balay, S., Buschelman, K., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc Web page (2009),
<http://www.mcs.anl.gov/petsc>
6. Karypis, G., Schloegel, K., Kumar, V.: PARMETIS Parallel Graph Partitioning and Sparse Matrix Ordering Library, University of Minnesota, Dept. of Computer Sci. and Eng. (1998)
7. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia (2000)
8. Ghanem, R., Spanos, P.: Stochastic Finite Element: A Spectral Approach. Springer, New York (1991)
9. Subber, W., Monajemi, H., Khalil, M., Sarkar, A.: A Scalable Parallel Uncertainty Analysis and Data Assimilation Framework Applied to some Hydraulic Problems. In: International Symposium on Uncertainties in Hydrologic and Hydraulic Modeling (2008)
10. Message Passing Interface Forum, <http://www.mpi-forum.org>
11. Keyes, D.: How scalable is Domain Decomposition in Practice? In: Proc. Int. Conf. Domain Decomposition Methods, pp. 286–297 (1998)
12. Dohrmann, C.: A Preconditioner for Substructuring Based on Constrained Energy Minimization. *SIAM Journal on Scientific Computing* 25, 246–258 (2003)
13. Farhat, C., Lesoinne, M., Pierson, K.: A Scalable Dual-Primal Domain Decomposition Method. *Numerical Linear Algebra with Application* 7, 687–714 (2000)

Interactive Data Mining on a CBEA Cluster

Sabine McConnell¹, David Patton², Richard Hurley¹,
Wilfred Blight¹, and Graeme Young¹

¹ Department of Computing and Information Systems

² Department of Physics and Astronomy

Trent University

1600 West Bank Drive, Peterborough,

ON Canada K9J7B8

{sabinemcconnell,dpatton,rhurley,wilfredblight,graemeyoung}@trentu.ca

Abstract. We present implementations of two data-mining algorithms on a CELL processor, and on a low-cost CBEA (CELL Broadband Engine Architecture) cluster using multiple PlayStation3 consoles. Typical batch-processing environments are often unsuitable for interactive data-mining processes that require repeated adjustments to parameters, pre-processing steps, and data, while contemporary desktops do not offer sufficient resources for the large datasets available today. Our implementations for the k Nearest Neighbour algorithm and the Decision Tree scale linearly with the number of samples in the training data and the number of processors, and demonstrate runtimes of under a minute for up to 500 000 samples.

Keywords: CELL Processor, Data Mining, High Performance Computing, Nearest Neighbour, Decision Tree.

1 Introduction

Data mining, defined as the extraction of novel information from large datasets, has established itself as a supplement to more traditional data analysis techniques in a large range of application domains. A coarse categorization divides data-mining techniques into *predictive* and *descriptive* approaches, with the goals to classify new data from a training set with class labels and to detect the inherent structure of the data, respectively. Popular techniques used for data mining are, for example, Artificial Neural Networks, Support Vector Machines, Decision Trees and Rule-Based classifiers. Data mining is a computationally expensive task, not only because of the interactive nature and the complexity of a large range of data-mining algorithms and techniques, but also because of the size of the datasets under investigation. To address mainly the latter problem, parallel implementations of individual data-mining techniques have been developed for a variety of distributed and shared memory architectures. However, even in contemporary settings, it is mandatory that run times be sufficiently short to allow for repeated adjustments of parameters, techniques, and data.

Notable exceptions (for example in astronomical applications) set aside, data mining is still typically applied in a desktop setting outside the computer science domain. This poses substantial limitations on the results that can be achieved. Even though parallel data-mining techniques are available, they do not necessarily match the interactive nature of data mining, in which individual researchers may apply different data-mining techniques to the same dataset, the same technique to different datasets, vary a set of parameters (for example the number of nodes in a hidden layer of an Artificial Neural Network), and so on, where results from a given application of a technique, often to a subset of the available data, will provide feedback for the next iteration. Therefore, a typical batch-processing high-performance environment is unsuitable, while the resources available through contemporary desktops are insufficient. With the goal of addressing this divergence, we investigate the possibilities and limitations for using a small PlayStation3®(PS3™) cluster for data mining in a desktop setting.

The remainder of this paper is organized as follows. Section 2 introduces the processor architecture and the cluster configuration. Section 3 discusses relevant work, including existing implementations of data mining algorithms on the CELL processor. Our algorithms and results are presented in Section 4. We then discuss considerations for data mining algorithms on this architecture in Section 5, followed by our conclusions in Section 6.

2 Background

2.1 Processor Architecture

In the past five years, the CELL Broadband Engine Architecture (CBEA) at the heart of the PS3, jointly developed by Sony, Toshiba, and IBM, has grown into a low-cost alternative for high-performance computing tasks, given certain restrictions with respect to main memory, latency, and precision of the floating-point calculations. This architecture, also used in its incarnation of the PowerXCell 8i for IBM's Roadrunner which tops the current list of supercomputers ranked according to their performance on the LINPACK benchmark, contains three main components.

The Power Processing Element (PPE), is a two-way hyperthreaded simplified PowerPC running at 3.2 GHz. The Synergistic Processing Elements (SPEs) consist of a vector processor, a private memory area of 256KB, communication channels, 128 128-bit registers, and a Memory Flow Controller. Each of the SPEs is single threaded, with a peak performance of 25 GFlops (single-precision). The Element Interconnect Bus, designed to improve bandwidth (theoretical maximum memory bandwidth of 25.2 GB/s) instead of latency connects the PPE and SPEs in a ring-like architecture. Efficient programming for this architecture typically requires using the PPE as resource manager only by offloading all data processing to the SPEs, requires explicit memory management by the applications and, for the CELL processor provided in the console, limits the application to single precision to take advantage of full pipelining.

The 6 SPEs in a single PS3™ CELL processor that are available to an application have a combined peak performance of 153.6 GFlops using fused multiplication-addition for single point precision. Provided that this power can be harnessed successfully in small-scale distributed environments for data mining, our small cluster of 5 consoles that requires a single power outlet has a theoretical single-precision floating point peak performance of just over 600 GFlops/s at a cost of under CDN \$2500. This is in addition to the modest floating-point performance of the PPEs. However, attaining the full potential of this fundamentally different architecture that connects the various components in a ring structure is non-trivial and requires a distinct programming paradigm, thus demanding substantial code changes even for existing distributed implementations.

2.2 Cluster Software and Hardware Configuration

In addition to typical issues such as the specific order in which the various components (Torque, Maui, etc.) had to be installed and configured, we also found that security programs in Linux interfered with the operations, to the point that SELinux and the built-in firewall needed to be disabled. In addition, there were issues relating to user access rights that would cause the Maui client to crash on the back end nodes. NFS was unreliable when used for sharing files between the consoles and the PC which we had planned to use as the front end. We were not able to compile MPI code on the PC and then execute that code on a PS3 cluster node. Eventually, we decided on a homogeneous cluster (with a PS3 as a front end) to resolve the cross-compilation issues, interconnected by a Fast Ethernet Switch (EE2400-SS). The final configuration of our cluster is as follows. OpenMPI, which includes assembly for PowerPC chips, is used as the message-passing interface. For development of the application, we used the IBM CELL Broadband Engine Software Development Kit (SDK) for Multicore Acceleration, Version 3.0, which includes compilers for C and C++. Torque (also known as Portable Batch System, or PBS), is used as the resource manager, which typically requires scripts to submit jobs to the queue but also supports interactive processing, combined with Maui for scheduling.

3 Related Work

Techniques used for the extraction of novel information from large datasets, are coarsely divided into predictive and descriptive approaches. Predictive approaches assign discrete (classification) or continuous (regression) class labels to unlabeled data, based on a model built from a labeled training set, while descriptive approaches aim to detect the inherent structure of the data, for example through clustering techniques. Data-mining techniques are used to provide answers to questions such as: *Is this object a star or a galaxy?*, *Which items are customers likely to buy together?*, or *Which of the connection attempts to our server are potentially malicious?* Popular data-mining techniques include for example Artificial Neural Networks, Support Vector Machines, Decision Trees and Rule-Based classifiers.

One of the characteristics of the application of data-mining techniques is that they are computationally intensive, arising from a number of factors that include the size of the datasets involved, the iterative nature of the data-mining process, the computational complexities of the algorithms, and the often empirical adjustment of parameters for a given technique, for example the number of nodes in a hidden layer of an Artificial Neural Network, or the number of clusters to be found by a clustering algorithm. In practice, a varying number of fundamentally different techniques will be applied multiple times to a given dataset, which will be repeatedly adjusted. This process is time-consuming, and, depending on the size of the dataset, often limits the choice of techniques that can be applied in a reasonable time frame.

The potential for scientific computing using the CELL architecture is discussed by Williams *et al.* [22] and demonstrated subsequently in various applications. Examples include the optimization of matrix multiplication by Kurzak *et al.* [18], the performance in biomolecular simulation by Fabritiis [8], the implementation of the Fast Fourier Transform by Chow *et al.* [5], and a discussion on the suitability of this architecture for relational database queries by Héman *et al.* [13]. In the context of data mining, various data-mining algorithms have been implemented on a single CBEA processor. Buehrer *et al.* [3] implement k -means clustering and the k Nearest Neighbours algorithm on a PS3 console with Fedora Core 5. The evaluation of this approach is based on varying values of k while the number of samples classified is fixed. Wang *et al.* [21] discuss a secure implementation of the k -means algorithm. Duan and Strey [9] report a speedup of up to 40 for k -means clustering, APriori for Association Mining, and a Radial Basis Function Network. Finally, Wyganowski [23] evaluate parallel implementations of a Multi-Layer Perceptron (an Artificial Neural Network) and a Support Vector Machine on a CELL processor. To our best knowledge, no implementations exist for Decision Trees on a single CELL processor, or any data-mining algorithms on a cluster of CELL processors.

However, a number of parallel implementations for the k Nearest Neighbour algorithm and Decision Trees have been proposed for distributed memory, shared memory, and hybrid architectures. Aparício *et al.* [2] discuss the implementation of the k Nearest-Neighbour classifier for a combination of three levels of parallelism using the Grid, MPI, and POSIX Threads. Focusing on the determination of the best value for k through cross validation, they submit multiple values of k to the Grid, using 10 MPI processes for the cross validation (one process for each partition), and finally POSIX threads for the computation of blocks of the MPI processes. For a dataset containing 20 attributes, in addition to the class label, Aparício *et al.* [2] report a speedup of 90 on a cluster of 3GHz PIII Xeon nodes, as compared to Weka, a Java-based data-mining tool. The reported speedup is divided into a speedup of 9.5 which is achieved by the allocation of cross-validation partitions to the 10 Biprocessor nodes, and a speedup of 1.5 from the use of four threads in the shared-memory environment. Note that the allocation of one process per partition limits the scalability of this approach, since cross-validation is typically performed on ten combinations of training and test set

data. Other parallel implementations of this technique include approaches by Gil-García et al. [10], Li [19], Jin [16], and Creecy et al. [7].

The building phase of decision trees can be performed utilizing data or task parallel techniques. In the data parallel, or synchronous, fashion, all available processors work on the same node. The decision tree is constructed one node at a time, either in a breadth-first or depth-first manner. In contrast, in the task parallel, or asynchronous approach, different processors are assigned to compute different subtrees of the complete decision tree. Typically, construction of the decision tree begins with a data parallel approach and switches to task parallel after a certain part of the decision tree is built. Parallel implementations for Decision Trees include ScalParC by Joshi et al. [17], Narlikar [20], Zaki et al. [24], Han et al. [12], and Andrade et al. [1].

4 Algorithms and Results

When used for classification, the Nearest-Neighbour approach assigns a new object to be classified to the majority of the classes of the object's nearest neighbours, generally based on the Euclidean distance. When a new object is to be classified, the algorithm compares it to all other objects and determines the k nearest neighbours, therefore requiring access to all of the data in order for each new sample. This algorithm is classified as a *lazy learner* since it does not build a model from the training data to be applied to new data, but rather stores all classified data.

Such a lazy learner is a prime candidate for implementation on a CELL cluster, since the SPEs in a given CELL processor are suitable for the distance calculation, requiring a nested loop and a limited number of branches (mainly caused by the need to maintain a list of k current closest neighbours) if each SPE has access to complete samples. In addition, the computation can be distributed over multiple processors by dividing either the data for which the classification is known, or the new samples, or both. Each datum is represented by its own vector, which corresponds to the typical layout of data in relational databases that contain samples in rows, and attribute values in columns. Note also that this algorithm is especially suitable for the short vectors supported by the SPEs (128 bits, which correspond for example to 16 character values, 4 single precision, or 2 double precision floating point values) because distances in higher dimensions tend to become meaningless.

For our parallel implementation, we extend the work by Buehrer et al. [3] which was restricted to 12 and 80 attributes and 100 000 classified samples, to determine the scalability of the approach to varying amounts of data, with or without class labels. In addition, we implement our technique on a cluster of PS3 consoles using MPI. In the latter case, the data to be classified as well as the classified data are distributed, with final classifications to be determined at the root. We evaluate the parallel implementations of our algorithms with artificial, balanced datasets containing 4 to 12 attributes and from 16384 to 262144 samples.

Algorithms 1 and 2 describe the high-level approach using double buffering, and the vectorized code at the SPEs, respectively. Initially, the data transfer commences for two distinct blocks of data. After completion of the first transfer, any necessary computation on the transferred data is performed at the same time as data transfer of the second block. After computation of the first block of data and completion of the second transfer, another data transfer is initiated and computation on the second block of data is performed at the same time as this transfer. Each SPE receives complete samples, and is responsible for determining the k Nearest Neighbours for a subset of the new data to be classified.

Algorithm 1. High-Level Description of Parallel k -NN for the CBEA Cluster using Double Buffering

Require: An integer k

- 1: Send copies of the training data to the different processors
 - 2: Distribute the data to be classified to the different processors
 - 3: **for all** processors **do**
 - 4: **for all** attributes **do**
 - 5: Send the starting address for the attribute to the next available SPE
 - 6: **while** The attribute is not fully processed **do**
 - 7: SPE initiates DMA transfer of the two next blocks of available data
 - 8: SPE waits for completion of the first transfer
 - 9: SPE computes on the data of the first block
 - 10: SPE initiates next transfer
 - 11: SPE waits for completion of the second transfer
 - 12: SPE computes on data of the second block
 - 13: **end while**
 - 14: **end for**
 - 15: **end for**
-

Algorithm 2. Determination of k -Nearest Neighbour at each SPE

- 1: **for all** Blocks of Data **do**
 - 2: **for all** Samples to be classified **do**
 - 3: **for all** Samples in the training data **do**
 - 4: Determine the Euclidean Distance between the datum to be classified and the neighbour
 - 5: **if** the distance to the training datum is smaller than the distance to the current k -th neighbour **then**
 - 6: Replace the current k -th neighbour with the training datum
 - 7: **end if**
 - 8: **end for**
 - 9: **end for**
 - 10: **end for**
-

Figure 1 shows the processing times for the parallel implementation of the k Nearest Neighbour algorithm on a single CELL processor for the classification of 32 768 samples, for 4 and 8 attributes.

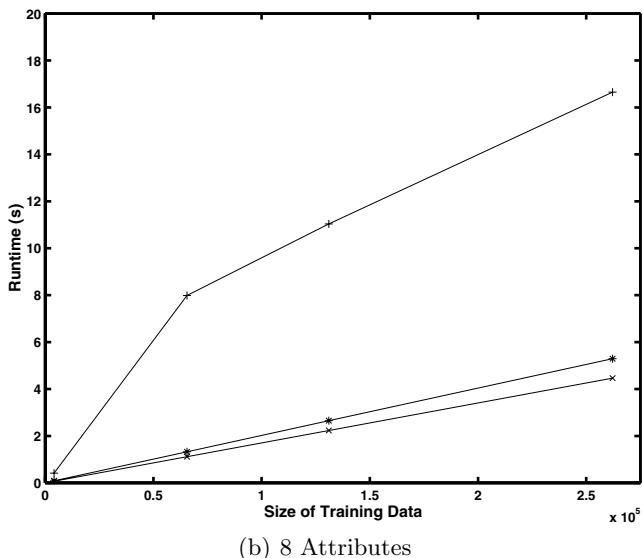
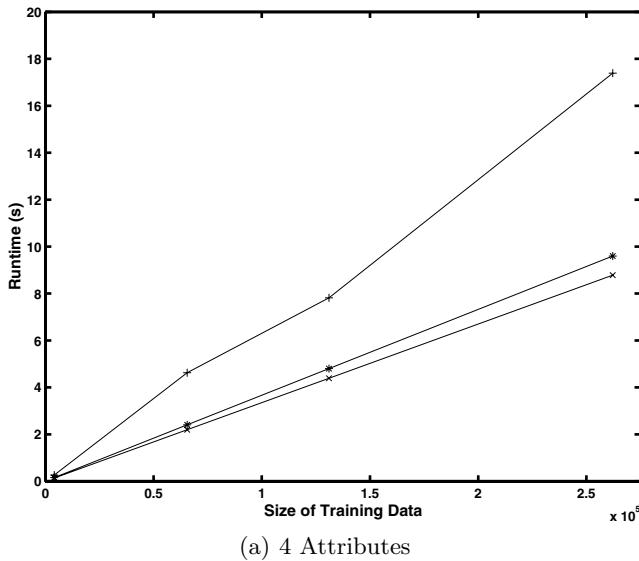


Fig. 1. Runtimes (in seconds) for the parallel k NN algorithm on a single CELL processor ($k = 7$), for increasing size of training data. Shown are the total time for the PPE(+), the total time for the SPE (*), and the vectorized portion of the code for the SPE (x).

The results indicate that the total processing time for the SPE and the vector processing time grow linearly with the size of the training data (up to roughly 260 000 samples). The vector capabilities of the SPEs are efficiently utilized as the time spent in the sequential part of the SPE code that maintains the list of the current k neighbours is small compared to the time spent in the vectorized code. It can also be seen that the overhead associated with the SPE threads is substantial, as the only task of the PPE in this implementation is the management of resources. This overhead is somewhat amortized with increasing number of attributes. Compared to the sequential implementation of the algorithm on the PPE only, the achieved speedups for a single CELL processor are 8.7 and 9.1 (32 768 new samples, 262 144 samples classified) for 4 and 8 attributes, respectively.

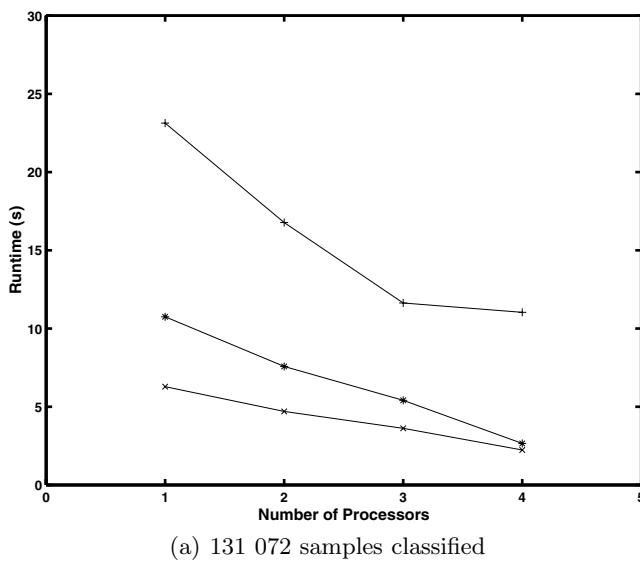
Figure 2 shows the runtimes for the implementation of the parallel k Nearest Neighbour algorithm on our cluster. Our implementation achieved speedups of 1.45, 2.02, and 2.8 for two, three, and four processors, respectively, as compared to the implementation on a single CELL processor. This indicates that the initially low speedup is caused by the start-up costs for the communication. In addition, the runtimes (under a minute) for the application of a lazy learner to fairly large datasets with a realistic number of attributes are low enough to support the typically interactive data-mining process.

4.1 Implementation of Decision Tree Algorithm

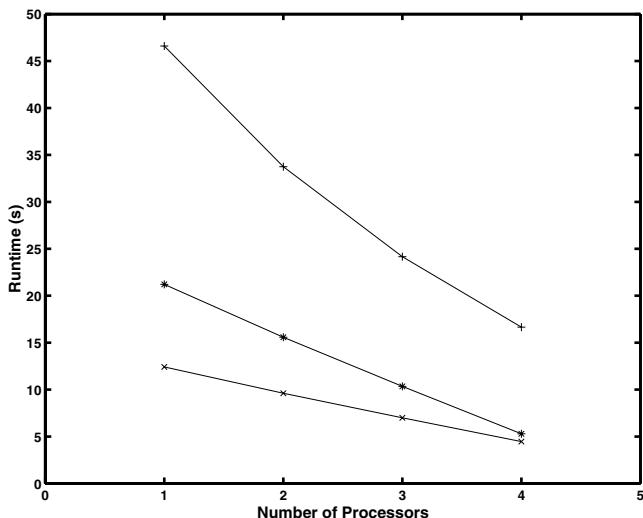
Decision tree algorithms build tree-like structures from the original data. The leaf nodes are associated with the target classes, while internal nodes are labeled with attributes and represent tests to be performed on the data. The outcome of the test is determined by the value of the attribute used to label the node. Initially, all the data is associated with the root. A splitting criterion such as the Gini index [11] or the Entropy is then used to separate the data, with the disjoint subsets associated now with the children of the root. The procedure is applied recursively until the nodes contain mainly samples of one class.

The building phase of the decision tree is typically followed by pruning to improve on the generalization error of the model, and a sample for which the class labels are unknown is classified by determining the leaf node in which it falls after following the path from the root that is determined by the sample's attribute values. Runtimes for the pruning phase and the subsequent classification of new data are orders of magnitude lower than those for construction of the tree. We therefore focused on the runtimes for the extraction of the tree from the classified data only. Since the Decision Tree algorithm accesses the data by attributes rather than by samples, the data is stored as Structures of Arrays. Again, we use double buffering to reduce the communication overhead, and the decision trees obtained by the parallel implementation are the same as those constructed with the sequential approach.

Algorithm 3 shows a high-level SPMD implementation of the Decision Tree construction process. At each individual processor, the SPEs are responsible for calculating the Gini indices of the attributes.



(a) 131 072 samples classified



(b) 262 144 samples classified

Fig. 2. Runtimes (in seconds) for the parallel k NN algorithm on the cluster (8 attributes, $k = 7$). Shown are the total time for the PPE (+), the total time for the SPE (*), and the time for the vectorized portion of the code for the SPE (x).

Algorithm 3. High-Level Description of Parallel Decision Tree Construction Process for the CBBA Cluster

- 1: Assign all data to the root in Processor 0
- 2: **for all** Attributes **do**
- 3: Next available SPE processes the attribute using multiple DMA transfers
- 4: **end for**
- 5: **while** termination criteria not satisfied **do**
- 6: PPE determines splitting attribute
- 7: split the data based on the values of the splitting attribute
- 8: send subsets of the data to the children for processing
- 9: process the data at the children
- 10: **end while**

Figure 3 shows a comparison of the runtimes for the sequential implementation on the PPE only and the parallel implementation using SPEs, for data containing categorical values only. Note that this occurs quite often in data-mining applications and that in this case the computation at the SPEs is restricted to counting, rather than distance calculation as in the case of the k Nearest Neighbour algorithm.

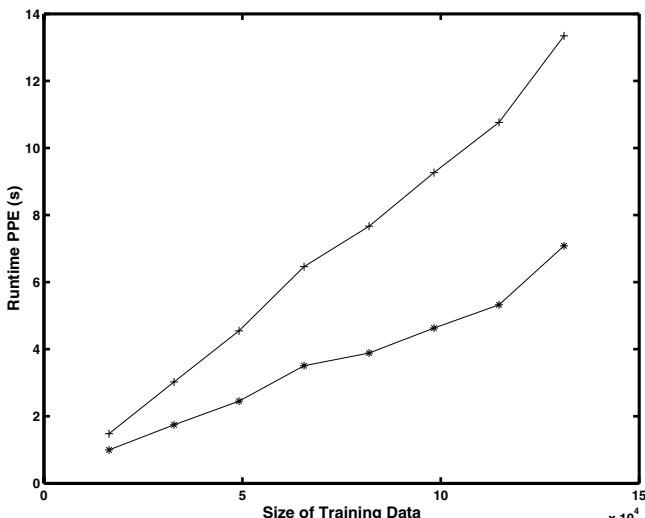


Fig. 3. Comparison of the runtimes (in seconds) for the sequential (PPE only, +) and parallel (PPE and SPEs, *) implementations of the Decision Tree algorithm (4 attributes) on a single CELL processor

An examination of the SPE timing file with `asmvis`¹ (a visualizer for the CELL assembly code) reveals the following. Disregarding code to create our timing measurements for the SPE, we find that in the Decision Tree implementation,

¹ Available at <http://www.alphaworks.ibm.com/tech/asmvis>

roughly 16% of clock cycles in the SPE are spent in stalls. These stalls occur mainly in the odd pipeline and are due to branching (roughly 10%) and to communication with the PPE via the channels. Stalls in the even pipeline are minimal, but do occur, for example, through the conversion of the computed Gini index to an Integer value for transmission through a mailbox, rather than DMA. The code displays strong vertical dependencies, minimizing opportunities for rearranging; and the majority of instructions are dual-issue, implying that both odd and even pipelines are occupied. Especially in the vectorized section of the code, we find a steep slope in the timing diagram, which indicates an efficient utilization of the architecture.

Therefore, the problem of the limited speedup must be rooted elsewhere, and may have some implications on the use of this architecture for data mining. For the k NN, the vectorized part of the SPE code has a minimal number of stalls in the odd pipeline that are due to the shuffles and rotates implemented to compute the count. The number of clock cycles is larger compared to those utilized in the implementation of the decision tree, and outside the vectorized section of the code, stalls tend to occur in both pipelines, mostly due to branches and less to communication with the SPE.

The poor speedup for the parallel implementation of the decision tree is due to two reasons: a) the overhead incurring at the PPE to split the current dataset into two, based on the selected splitting criterion and b) the fact that the size of the data decreases roughly by half at each iteration of the algorithm, which means that the communication overhead for the lower levels of the tree (for which the number of nodes grow as a factor of two with each level) cannot be as easily amortized by the speedup gained from using the SPEs during the construction of the upper levels of the tree.

Figure 4 shows the runtimes for the building phase of the parallel implementation of the Decision Tree on our cluster, for increasing training data sizes. The speedup increases for increasing number of attributes, and for increasing size of the training data. For example, for 65 536 samples and 12 attributes, the parallel implementation achieves a speedup of 6.24 as compared to the sequential implementation on the PPE only.

5 Consideration of the Characteristics of the Architecture for Data Mining Techniques

The CELL processor supports a wide range of programming models and therefore a wide range of applications in, for example, image processing, digital signal processing, or biomolecular simulations. Typically, the PPE is used as resource manager only by offloading all computation to the SPE. Choices for the programming model include the Multistage Pipeline, Parallel Stages, and the Services model. To date, the most popular approach is the Parallel Stages model, in which the same algorithm is applied to different parts of the data. In terms of data mining, this model is not always the most appropriate. For example, the Decision Tree algorithm may be more suitable for the multistage pipeline, at least for the categorical data investigated here. In addition, and depending on the data-mining technique, it should

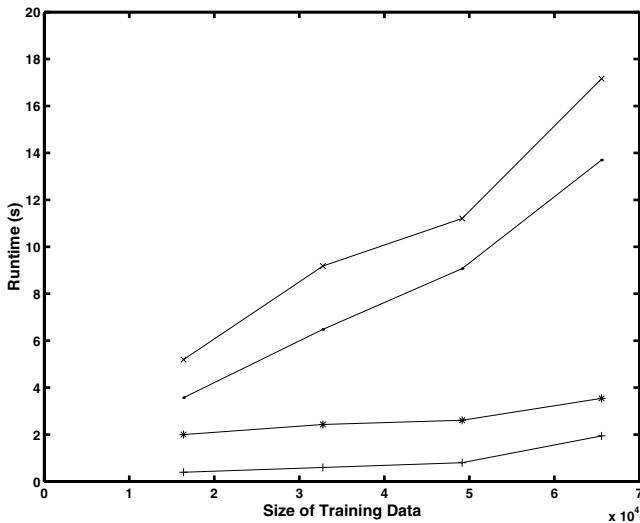


Fig. 4. Runtimes (in seconds) for the parallel Decision Tree algorithm on the cluster of four CELL processors for six(+), eight(*), ten(.), and twelve attributes(x)

be useful to include the vector-processing capabilities of the PPE, along with its second thread, for implementations of data-mining techniques, rather than offloading all computation to the SPE. In particular, certain pre-processing requirements or determination of overall winners such as for the k Nearest Neighbour approach could be handled by the PPE.

The bandwidth available for communication between processor elements will be used most efficiently when SPEs communicate with each other rather than with the main memory [4]. This will not always be the case for the most efficient implementation of a data-mining algorithm on this architecture. An example is the Neural Network algorithm, for which we are currently implementing a layered approach in which the communication requirement decreases as we proceed through the layers. This also implies that DMA lists, which are critical for efficient access to main memory for large datasets, cannot be utilized for implementations of all data-mining algorithms.

In the CELL processor for the PS3TM console, only single-precision arithmetic is fully pipelined. Furthermore, the single-precision floating point computation in the SPEs is not fully compliant with IEEE Standard 754; for example, there is an extended range of normalized numbers and truncation of the numbers. However, this does not impose significant restrictions on the application of data-mining techniques with this architecture, since typically the data contains some level of measurement errors since they are a by-product of some other process. Furthermore, the small size of the unified local store of the SPEs does not pose a restriction on any single implementation of a data-mining technique which typically produces fairly small executables.

6 Conclusions

We present implementations of the k Nearest Neighbour and Decision Tree algorithms on a CELL processor, and on a low-cost cluster using multiple PlayStation3 consoles. Previous work was restricted to the implementation of the k Nearest Neighbour algorithm on a single Cell processor, while decision tree algorithms were not implemented on this architecture. The total runtime for PPE and SPE as well as the vector processing time for both implementations scale linearly with the number of samples in the training data and the number of processors, for both techniques. The observed runtimes are low enough for the medium size samples to support interactive data mining on the cluster.

However, we also observed a steep learning curve necessary to take advantage of the characteristics of the architecture, and to set up the cluster, in combination with a low speedup for the Decision Tree implementation on a single CELL processor that is caused by the low computation to communication ratio. In addition to identifying further opportunities for implementations of decision trees, we currently plan to focus on implementations of Artificial Neural Networks, Bayesian techniques, and Support Vector Machines on this architecture. Given that a large variety of programming models are supported by the CELL processor and that data characteristics, access patterns, and computational complexities vary substantially among data-mining techniques and even application areas, it is unlikely that a single programming model will be sufficient for the efficient implementation of data-mining techniques on this architecture.

Acknowledgments. This work was made possible by a Fellowship Award from the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca).

References

1. Andrade, H., Kurc, T., Sussman, A., Saltz, J.: Decision Tree Construction for Data Mining on Clusters of Shared-Memory Multiprocessors. In: Proceedings of International Workshop on High Performance Data Mining, HPDM 2003 (2003)
2. Aparicio, G., Blanquer, I., Hernández, V.: A Parallel Implementation of the K Nearest Neighbours Classifier in Three Levels: Threads, MPI Processes and the Grid. In: Daydé, M., Palma, J.M.L.M., Coutinho, Á.L.G.A., Pacitti, E., Lopes, J.C. (eds.) VECPAR 2006. LNCS, vol. 4395, pp. 225–235. Springer, Heidelberg (2007)
3. Buehere, G., Parthasarathy, S., Goyder, M.: Data Mining on the Cell Broadband Engine. In: Proceedings of the 22nd Annual International Conference on Supercomputing, pp. 26–35. ACM, New York (2008)
4. Buttari, A., Luszczek, P., Kurzak, J., Dongarra, J., Bosilca, G.: SCOP3. A Rough Guide to Scientific Computing on the PlayStation3. Technical Report UT-CS-07-595, Innovative Computing Laboratory, University of Tennessee Knoxville (2007)
5. Chow, A., Fossum, G., Brokenshire, D.: A Programming Example: Large FFT on the Cell Broadband Engine, IBM (2005)

6. Cover, T.M., Hart, P.E.: Nearest Neighbour Pattern Recognition. *IEEE Transactions on Information Theory* 13(1) (1967)
7. Creecy, R.H., Masand, B.M.H., Smith, S.J., Waltz, D.L.: Trading MIPS and Memory for Knowledge Engineering. *Communications of the ACM* 35(8), 48–63 (1992)
8. DeFabritiis, G.: Performance of the Cell Processor for Biomolecular Simulations. *Computer Physics Communications* 176(11-12), 660–664 (2007)
9. Duan, R., Strej, A.: Data Mining Algorithms on the Cell Broadband Engine. In: Luque, E., Margalef, T., Benítez, D. (eds.) *Euro-Par 2008. LNCS*, vol. 5168, pp. 665–675. Springer, Heidelberg (2008)
10. Gil-García, R.J., Badía-Contelles, J.M., Pons-Porrata, A.: Parallel Nearest Neighbour Algorithms for Text Categorization. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) *Euro-Par 2007. LNCS*, vol. 4641, pp. 328–337. Springer, Heidelberg (2007)
11. Gini, C.: Measurement of Inequality of Incomes. *The Economic Journal* 31, 124–126 (1921)
12. Han, E., Srivastava, A., Kumar, V.: Parallel Formulation of Inductive Classification Learning Algorithm. Technical Report 96-040, Department of Computer and Information Sciences, University of Minnesota (1996)
13. Héman, S., Nes, N., Zukowski, M., Boncz, P.: Vectorized Data Processing on the Cell Broadband Engine. In: Proceedings of the 3rd International Workshop on Data Management on New Hardware, Beijing, China (2007)
14. Hong, W., Takizawa, H., Kobayashi, H.: A Performance Study of Secure Data Mining on the Cell Processor. In: *8th IEEE International Symposium on Cluster Computing and the Grid*, pp. 633–638. IEEE Press, New York (2008)
15. Programming the Cell Broadband Engine Architecture: Examples and Best Practices, <http://www.redbooks.ibm.com/abstracts/sg247575.html>
16. Jin, R., Yang, G., Agrawal, G.: Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming Interface and Performance. *IEEE Transactions on Knowledge and Data Engineering* 17, 71–89 (2005)
17. Joshi, M., Karypis, G., Kumar, V.: ScalParC: A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets. In: *Proceedings of the 11th International Parallel Processing Symposium*. IEEE Computer Society Press, Los Alamitos (1998)
18. Kurzak, J., Alvaro, W., Dongarra, J.: Optimizing Matrix Multiplication for a Short-Vector SIMD Architecture - CELL processor. *Parallel Computing* 35, 138–150 (2009)
19. Li, X.: Nearest Neighbour Classification on two types of SIMD machines. *Parallel Computing* 17, 381–407 (1991)
20. Narlikar, G.: A Parallel, Multithreaded Decision Tree Builder. CMU-CS-98-184. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA (1998)
21. Wang, H., Takizawa, H., Kobayashi, H.: A Performance Study of Secure Data Mining on the Cell Processor. In: *International Symposium on Cluster Computing and the Grid*, pp. 633–638 (2008)
22. Williams, S., Shalf, J., Oliker, L., Kamil, S., Husbands, P., Yelick, K.: The Potential of the Cell Processor for Scientific Computing. In: *Proceedings of the 3rd Conference on Computing Frontiers*, pp. 9–20. ACM, New York (2006)
23. Wyganowski, M.: Classification Algorithms on the Cell Processor. MSc. Thesis, Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY (2008)
24. Zaki, M., Ho, C., Agrawal, R.: Parallel Classification on SMP Systems. In: *The 1st Workshop on High Performance Data Mining (in conjunction with IPPS 1998)*, Orlando, FL, USA (1998)

Service-Oriented Grid Computing for SAFORAH

Ashok Agarwal³, Patrick Armstrong³, Andre Charbonneau⁴, Hao Chen², Ronald J. Desmarais³, Ian Gable³, David G. Goodenough^{1,2}, Aimin Guan², Roger Impey⁴, Belaid Moa¹, Wayne Podaima⁴, and Randall Sobie^{3,5}

¹ Department of Computer Science, University of Victoria, Victoria, BC

² Canadian Forest Service, Natural Resources Canada (NRCan)

³ Department of Physics and Astronomy, University of Victoria, Victoria, BC

⁴ Information Management Services Branch, National Research Council Canada, Ottawa, Ontario

⁵ Institut for Particle Physics, Canada

Abstract. The SAFORAH project (System of Agents for Forest Observation Research with Advanced Hierarchies) was created to coordinate and streamline the archiving and sharing of large geospatial data sets between various research groups within the Canadian Forest Service, the University of Victoria, and various other academic and government partners. Recently, it has become apparent that the availability of image processing services would improve the utility of the SAFORAH system. We describe a project to integrate SAFORAH with a computational grid using the Globus middleware. We outline a modular design that will allow us to incorporate new components as well as enhance the long-term sustainability of the project. We will describe the status of this project showing how it will add a new capability to the SAFORAH forestry project giving researchers a powerful tool for environmental and forestry research.

Keywords: Forestry, remote sensing, computing, grids

1 Introduction

Research projects are building very sophisticated instruments to study the world we live in and the universe surrounding us. Telescopes, satellites, accelerators, and ocean laboratories are collecting vast amounts of data for basic and applied research. Dealing with such large data sets is a challenge and many projects are adopting emerging data grid techniques. The Canadian Forest Service's SAFORAH (System of Agents for Forest Observation Research with Advanced Hierarchies) is an example of a project that collects data from a variety of land and spaced-borne remote sensing sources [1]. SAFORAH was created to coordinate and streamline the archiving and sharing of large geospatial data sets between various research groups within the Canadian Forest Service, and various other academic and government partners. Currently, only the raw or unprocessed images and data sets are made available on the SAFORAH data grid. Recently it has become apparent that providing researchers with processed images would significantly enhance SAFORAH. We describe a

project to add the ability to process data in SAFORAH to create new information products using a service-oriented computational grid [2] based on the Globus Toolkit Version 4 (GT4) middleware [3].

The new functionality in SAFORAH is supported by integrating the GT4 computational grid provided by the Gavia Project as shown in Figure 1. The integration is done using a grid service with which SAFORAH user interface and third party applications can interact. The Gavia Project is a developmental computational grid using a service-oriented architecture initiated in 2006. It is based on GT4 and uses a metascheduler based on Condor-G [4]. The Gavia Project provides transparent access from the SAFORAH system to a set of distributed computational facilities in Canada. The architecture of the system also allows the use of other metaschedulers such as Gridway [10] which are currently under development by other groups.

Gavia integrates the task brokering services provided by the Condor-G package with GT4. Prior to Gavia, the project team established the GridX1 Project, a computational grid in Canada using older versions of Condor-G and GT2 software. GridX1 was used to run production simulation jobs for particle physics experiments [5].

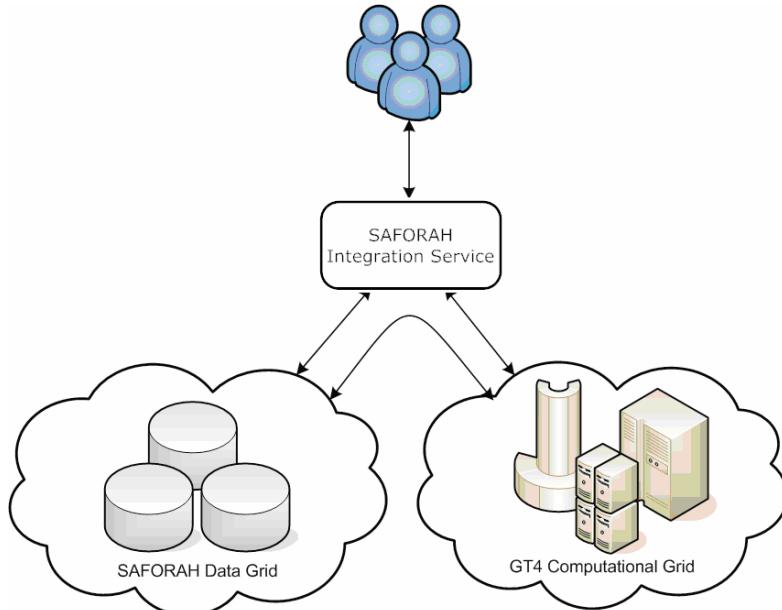


Fig. 1. High-level overview showing the integration of a GT4 computational grid into the SAFORAH system. The SAFORAH data grid and the GT4 computational grid are pre-existing systems that are united through a service.

2 Overview of the SAFORAH System

SAFORAH was originally created to coordinate and streamline the archiving and sharing of large geospatial data sets between various research groups within the

Canadian Forest Service, and various other academic and government partners within a secure framework. SAFORAH makes optimum use of distributed data storage to enhance collaboration, streamline research workflows and increase the return on the Canadian government's investment in Earth Observation (EO) data by freeing researchers from focusing on data storage and distribution issues.

The current SAFORAH data grid provides significant benefits to the participating organizations and partners who work on the large national projects to monitor and report on the state of Canada's forests as required by the Canadian Parliament and international agreements. Land cover mapping is often the primary source for determining current status of an area and is used as a baseline for future changes. Increasingly, remote sensing is being turned to as a timely and accurate data source for land cover mapping. Many provincial and territorial mapping agencies have recognized the importance of Landsat remotely-sensed land cover mapping. They have started utilizing remote sensing information products in SAFORAH to improve their decision-making processes on sustainable development and environmental assessment.

SAFORAH presents the user with a web portal to the data distributed in the various storage facilities across multiple organizations and government agencies. Users, depending upon their access privileges, can download or upload any predefined EO imagery from anywhere on the data grid. Currently, four Canadian Forest Service centres (Victoria, Cornerbrook, Edmonton, and Québec City), Environment Canada's Canadian Wildlife Service in Ottawa, the storage facility at University of Victoria and the University of Victoria Geography Department are operationally connected to the SAFORAH data grid. Two sites at the Canadian Space Agency (CSA) and the University of Victoria's Computer Science Department are also linked to SAFORAH. The metadata is stored in either the Catalogue and User Data Ordering System (CUDOS) or the Open Geospatial Consortium (OGC) [6] Catalogue Service for Web (CS/W) for catalogue, data archiving and ordering.

SAFORAH requires new computing and storage resources to handle both the growing demands for remote sensing data from hyperspectral, multispectral and radar satellites and the demands for processing capabilities for derived information products. Grid computing and high-speed networking offers a solution to these challenges by allowing grid users to access not only heterogeneous storage facilities but also processing resources distributed across administrative domains.

SAFORAH offers two main components for users to interact with the data grid shown in Figure 2: the Catalogue and User Data Ordering System (CUDOS) from MacDonald, Dettwiler and Associates (MDA), and the more recent web-services interface based on the OGC standards and specifications. The main difference between CUDOS and OGC is that CUDOS allows users to access a concatenated granule in a zipped format containing the complete image and associated files. OGC allows other applications and services to access EO data, stored as individual channels or bands for each EO image, such as a 400-band hyperspectral image. Currently, a simple HTTP interface is built so that SAFORAH users can access OGC services and request registered map images or accessing geospatial coverage objects. The two systems, CUDOS and OGC, provide access to complementary data sets.

CUDOS, which conforms to the US metadata standard established by the Federal Geographic Data Committee [7] (FGDC), uses pre-Web-service components of Globus Toolkit 4 (GT4) and has its own security mechanisms for user authentication and access controls to SAFORAH. The connection between CUDOS and the SAFORAH data grid is authenticated by using Grid Canada credentials.

The OGC services use GT4 web services. The implementation enhances information interchange with other geospatial information systems in the Canadian Geospatial Data Infrastructure (CGDI), such as the CFS National Forest Information System (NFIS). The services include the Catalogue Service, Coverage Service, Web Map Service, and other supporting grid services. Other OGC clients or GIS-based information systems may also gain access to these grid-enabled services to obtain EO data through the standard OGC Web service portals.

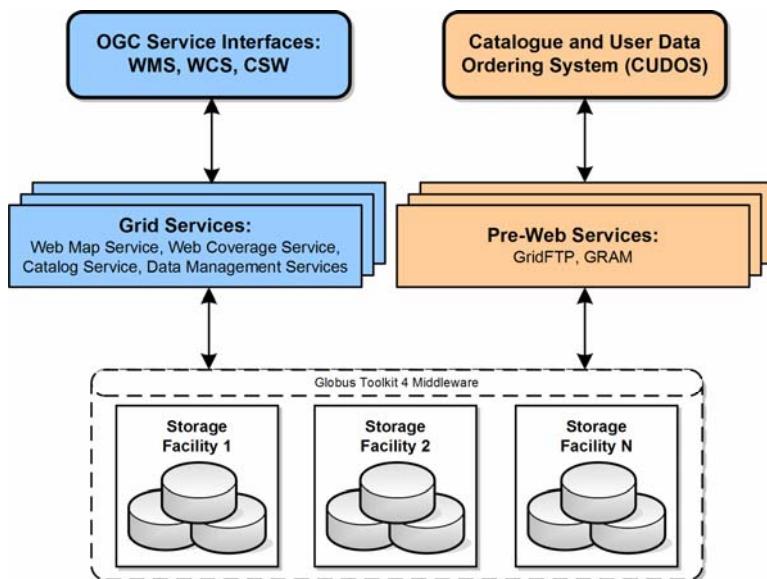


Fig. 2. Architecture of the SAFORAH system showing the two components CUDOS and the OGC interface. CUDOS uses pre-WS components while the OGC interface utilizes the WS components in the Globus Toolkit V4.

3 Overview of the Computational Grid Service

The OGC services of SAFORAH are built using the Web Services Resource Framework [8] (WSRF) standard, which is the same standard used by GT4. It was therefore a logical choice to integrate it with a GT4 computational grid for SAFORAH's computation requirements.

Members of the project team have constructed a small GT4 computational grid running on a number of Canadian computing resources. The components of a GT4 grid are shown in Figure 3. The two key components are the metascheduler and registry service. The role of the metascheduler is to broker access to the distributed

computational facilities, which schedule jobs to individual worker nodes: the metascheduler acts as a scheduler of schedulers. As a central grid service, the metascheduler provides the functionality to submit, monitor, and manage jobs across the facilities. The registry is a second grid service which stores and accesses resource advertisements from the computing facilities. The type and number of resources that are available are included in each advertisement. Additional services to manage data and security are also components of the grid.

The GT4 computational grid was designed in a modular fashion to allow it to use components from other projects. This group has previously developed metascheduler and registry services based on Condor-G. The metascheduler is called Gavia, and is a Globus incubator project [9].

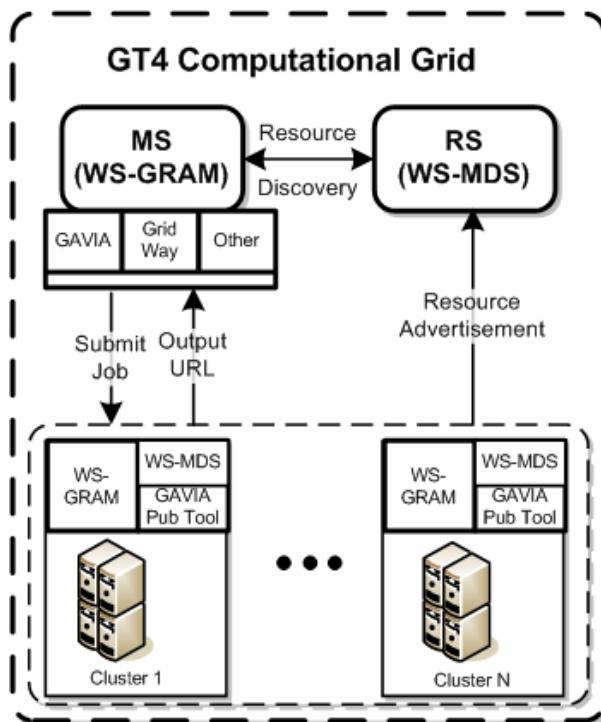


Fig. 3. An overview of the components of a GT4 computational grid showing the metascheduler service (MS) and the registry service (RS). Each cluster publishes its status to the registry, which is used by the metascheduler to schedule jobs.

Gavia is built using WSRF, a product of the Open Grid Forum, and is implemented using GT4. The WSRF includes standards for job management and for resource registry: Web Services Grid Resource Allocation and Management (WS-GRAM) is a standard which was created to allow the management of job execution on grid resources, and Web Services Monitoring and Discovery System (WS-MDS), a standard for the advertisement and discovery of grid resources. Since WS-MDS is

implemented as part of GT4, the Gavia registry is simply a deployment of this technology. However, during the initial development, there was no mature and stable WS-GRAM-enabled metascheduler. As such, a metascheduler was developed using Condor-G at the core for making allocation decisions. The Gavia metascheduler was constructed by developing a WS-GRAM interface to Condor-G, as well as a mechanism to discover available resources in the registry. This system allows users or automated clients to submit jobs to a single service, which in turn performs an allocation decision and resubmits the job to the selected facility. By providing transparent access to many geographically-distributed facilities, users are able to focus on their applications and research, rather than having the burden of managing their executions at many facilities at once.

There is an alternative metascheduler developed by the GridWay project [10]. This project's computational grid can operate using either the Gavia or GridWay metascheduler.

4 SAFORAH Computational Grid Service

Figure 4 shows the integrated SAFORAH and computational grid systems. The existing SAFORAH system is shown on the left, and the GT4 computational grid on the right. A

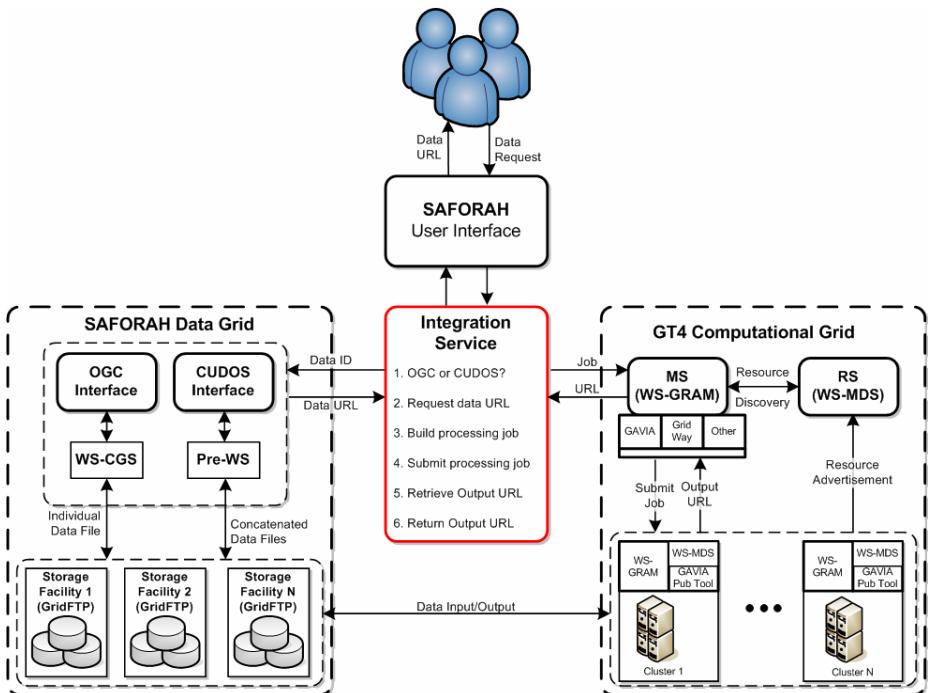


Fig. 4. A detailed figure showing the integration of the SAFORAH data grid with a GT4 computational grid. The researcher access the system via SAFORAH user interface and the Integration Service (IS) retrieves the data location from SAFORAH, runs the requested application on the grid and returns the links for the output to the researcher.

new component, the Integration Service (IS), is shown in the center. The purpose of this service is to coordinate the interaction between the existing data grid and the new computational grid. The decisions made by the IS are listed within the red box. The IS is implemented as a grid service that allows web portals and other systems to use both grids in a transparent manner. Note that the user is not involved in the data transfer between the two grids. Using SAFORAH web portal, the user specifies only the area of interest and the product to generate and the IS handles the rest.

We give a brief overview of how the system will respond to a user request: The user submits a request via the SAFORAH web portal. If the request is for existing data, the system then emails the user a link to the data. If the request is for computed data, the request is sent to the IS, which then locates the input data, generates the necessary scripts, creates the job description and sends the job to the metascheduler. The metascheduler finds the appropriate resource after querying the registry service and submits the job to one of the clusters, which then internally submits it to one of its worker nodes. After the job is completed, the data is written to a SAFORAH node and a link is returned to the user via the Integration Service or email.

5 Status of the Project

The project is ending its first year of development as part of a two-year project. A beta prototype for the system has been implemented and tested. The IS of the prototype is implemented based on WSRF using GT4 and can interact with other services and third party applications. The existing SAFORAH web portal has been

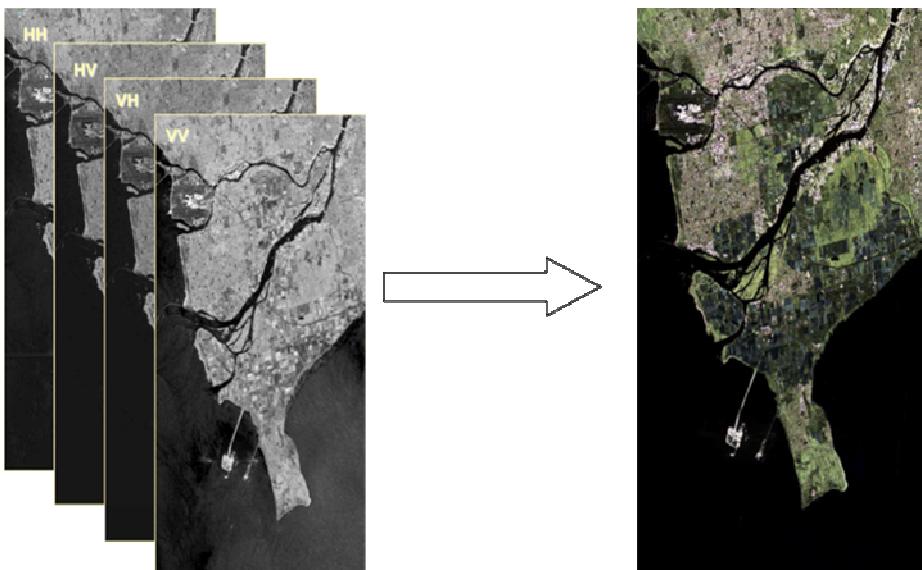


Fig. 5. SCM takes four polarization channels of Radarsat-2 FQ15 data, and produces an HSV land cover product image. The image shown highlights the Vancouver area.

extended to interact with the IS so that users can generate products, which we call virtual products, on the fly using the GT4 computational grid. The transfer of data between the SAFORAH data grid and the GT4 computational grid is performed on high speed links (1 GB/s and up). This makes the system efficient in handling large data sets. Currently, authorized users have access to two virtual products: a hyperspectral calibration product (force-fit) and Shane Cloude RADAR decomposition method (SCM). For more information about these two products, the reader may refer to [11] and [12, 13], respectively. The force-fit product is a very lightweight product and takes only several minutes to run. It allows us to test the system and check its state quickly. SCM takes much longer to complete (around one-half of an hour for a 300MB input data) and allows the users to generate a real forestry product. Figure 5 shows an example of the virtual product computed using SCM.

6 Conclusion

SAFORAH has been limited to providing its users with access to pre-existing data sets and has relied upon users themselves to provide the computational resources, algorithms, and experience to produce derived information products from the raw data. The addition of the SAFORAH computational grid service to SAFORAH allows users to obtain derived information products without having to download, store, and process the raw data themselves. The initial implementation of the service has shown it possible to run hyperspectral calibration methods and the Shane Cloude RADAR decomposition methods within a GT4 computational grid, while pulling data from the existing SAFORAH Data Grid.

Acknowledgements

We thank Natural Resources Canada, the University of Victoria, National Research Council Canada, Natural Sciences and Engineering Research Council of Canada, CANARIE Inc. and the Canadian Space Agency for their financial support. We are very grateful for the technical support of the Center for Spatial Information Science and Systems of George Mason University for the grid SOA development. We appreciate the high-bandwidth connectivity provided by Shared Services BC and the Radarsat-2 data provided by MDA.

References

- [1] Goodenough, D.G., Chen, H., Di, L., Guan, A., Wei, Y., Dyk, A., Hobart, G.: Grid-enabled OGC Environment for EO Data and Services in Support of Canada's Forest Applications. In: Proceedings of IGARSS 2007. IEEE International, July 2007, vol. 4773 (2007)
- [2] Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. In: Jin, H., Reed, D., Jiang, W. (eds.) NPC 2005. LNCS, vol. 3779, pp. 2–13. Springer, Heidelberg (2005)

- [3] The Globus Alliance, The Globus Toolkit, <http://www.globus.org>
- [4] Condor – high throughput computing, <http://www.cs.wisc.edu/condor>
- [5] Agarwal, A., et al.: GridX1: A Canadian computational grid. Future Generation Computer Systems 23(5), 680–687 (2007)
- [6] Open Geospatial Consortium, Inc., <http://www.opengeospatial.org/>
- [7] The Federal Geographic Data Committee, <http://www.fgdc.gov/>
- [8] The WS-Resource Framework, <http://www.globus.org/wsrf/>
- [9] The Gavia Project, <http://dev.globus.org/wiki/Incubator/Gavia-MS>
- [10] GridWay Metascheduler - Metascheduling Technologies for the Grid,
<http://www.gridway.org/doku.php>
- [11] Goodenough, D.G., Dyk, A., Hobart, G., Chen, H.: Forest Information Products from Hyperspectral Data - Victoria and Hoquiam Test Sites. In: Proc. IGARSS 2007, Barcelona, Spain, pp. 1532–1536 (2007)
- [12] Cloude, S.R.: Radar target decomposition theorems. Inst. Elect. Eng. Electron. Lett. 21(1), 22–24 (1985)
- [13] Cloude, S.R., Chen, E., Li, Z., Tian, X., Pang, Y., Li, S., Pottier, E., Ferro-Famil, L., Neumann, M., Hong, W., Cao, F., Wang, Y.P., Papathanassiou, K.P.: Forest Structure Estimation Using Space Borne Polarimetric Radar: An ALOS-PALSAR Case Study. ESA Dragon Project (2007)

IO Pattern Characterization of HPC Applications

Jeffrey Layton

Dell, Inc., One Dell Way, Round Rock, Texas
jeffrey_layton@dell.com

Abstract. The HPC world is waking up to the importance of storage as an integral part of an HPC solution. Many HPC sites are realizing that HPC class storage coupled with classic enterprise features such as snapshots, replication, disaster recovery, and ILM (Integrated Life Cycle Management), can result in the maximum performance and uptime for HPC applications. However, the biggest problem that HPC centers face is that they don't understand how their applications perform IO (i.e. IO pattern). This means that they can't define meaningful requirements for HPC storage. Knowledge of the IO patterns allows better definition of requirements including both time and money. This paper presents a technique for obtaining IO information and analyzing it to present IO pattern information. The technique is independent of the application.

Keywords: HPC, File System, WRF, strace.

1 Introduction

Fundamentally, HPC storage is not thought of as a “sexy” aspect of HPC. There is no “Top500” for HPC storage that can easily show funding agencies where and how their money has been spent. HPC systems have typically focused on the computing and network aspects of the system with storage being relegated to the status of a “necessary evil” (something of an afterthought in the design of the system).

The result is an HPC storage system that is woefully mismatched to the true IO requirements. Many times the HPC Storage is the bottleneck, but at the same time, there are many storage systems that are over-specified with too much performance. In addition to badly mismatched storage systems is a lack of knowledge of where the IO data requirements are headed over time. The end result is something of a mess for the HPC centers. In the author’s experience, on average, HPC Storage accounts for about 15-25% of the total system cost, but accounts for about 80-90% of the problems.

At that same time the storage capacity requirements in HPCC have exploded. With this explosion have come management problems. For example, backing-up hundreds of Terabytes is not practical any more, at least not in the traditional sense of a weekly full backup with daily incremental backups. Moreover, not all of the data needs to be on-line all of the time. Issues such as these have led to the use of classic enterprise features such as snapshots, replication, and ILM (Integrated Life Cycle Management) to help alleviate these problems.

Making the problem worse is the fantastic price/performance characteristics of clusters that have given users the ability to generate enormous amounts of data. This

has led to this two-pronged problem of having good IO performance so that storage is not a bottleneck or having too much IO performance and wasting money, coupled with a huge increase in capacity and its associated management problems. This is forcing HPC centers to focus on the design of storage systems for the applications they are supporting.

One common question or concern is how to get started designing an HPC storage solution. The next logical question one should ask is, “how can I develop the performance requirements for my HPC Storage system?” This question is answered in much the same way as the same question on the compute side, “what are the IO requirements of the applications?” Unfortunately, in the author’s experience there are very, very few application developers than can answer this question and even fewer that can articulate the answer in terms of storage requirements. This paper will introduce, or re-introduce, a simple technique for gathering and examining IO information of your applications.

2 Profiling IO Patterns

There have been past efforts at gathering information about the IO pattern of an application. One of the simplest techniques is to instrument the code. This can be rather laborious and complex depending upon the approach, and it could introduce a number of complications increasing code size and perhaps interfering with any measurements of the impact of IO on the overall application performance.

Another approach is to track “traces” of the application IO’s patterns as the application is running. While it sounds difficult, this is actually very easy to accomplish simple using the strace tool commonly found in most *nix systems.

Strace is more commonly used for debugging applications, but it is seldom realized that on **ix systems, the IO is performed using library functions. Consequently, strace can trace these function calls providing information on how the application is performing IO. In essence, IO patterns of the application are captured. Moreover, strace also includes the option of microsecond timing to the functions. Combining the details of the O function with the execution time, the throughput performance can be computed.

The benefits of using strace are fairly obvious – the system IO functions details can be captured including the IO operation (i.e., what function was called), how much data was in the IO operation, how long the function took to perform the operation (this is a function of the storage system used during the tests), etc. In addition, no modifications to the application are needed to use strace. Moreover, for applications that are sensitive or data sets that are proprietary, it is almost impossible to use the strace output to reconstruct either the data or the application. This is also true for ISV applications to which users do not have access to source.

However, for applications with reasonably long run times, strace can create an enormous amount of data. This creates the problem of having to sort through the data to discern the IO patterns. This paper will present a relatively simple technique of characterizing the IO pattern of an application that is independent of the application itself and can be used by the user (i.e. no root access is required). The techniques rely

on the common **ix tool, strace. Using strace and a tool developed by the author called strace_analyzer, an application can be analyzed for the following information:

- Read/Write throughput (as a function of time during the application’s run)
- Read/Write data sizes per function call
- IOPS (read, write, and total – also as a function of time)
- Lseek frequency
- Average read and write function call size (how many bytes per function call)
- Slowest IO function (read, write, lseek, open, close, etc.)
- Amount of time spent performing IO
- Plots of write and read functions as a function of time

From this information one can then begin to understand how applications are performing IO (IO patterns). Perhaps more importantly, it can provide information on application characteristics that can be used to design an adequate storage system (i.e. requirements).

3 Running strace_analyzer

The first step in running the strace_analyzer is to produce the strace output. This is fairly easy to accomplish by using the strace utility that comes with virtually every version of *nix. The most important aspect of using strace is to use the “-tt” option to ensure microsecond timing accuracy. Perhaps equally important, particularly for HPC, is how you can use this tool with MPI applications.

If you use strace arbitrarily such as,

```
#strace mpiexec -np 4 <application>
```

you are actually running strace against mpiexec and not against the application as you desire. So you have to make a few changes to your script so that you run strace against the application. There is an article on-line at Ref. [6] that gives the details. Basically, you have to create a script just for running the application itself that sends the strace output to an individual file. Then you modify your job script to call that “application script”. Your job script looks like the following:

```
#!/bin/bash
mpiexec -machinefile ./MACHINEFILE -np 4 <path-to-script>/code1.sh <code-options>
```

and your application script looks like the following:

```
#!/bin/bash
/usr/bin/strace -tt -o /tmp/strace.out.$$ <path-to-code>/<executable> $@
```

The reference contains some details as well as a simple example of how you to write these scripts.

4 Test System and Test Application

To illustrate what can be done with strace and strace_analyzer, a simple example was used. The application is called WRF (Weather Research and Forecasting). [1,2] It is a multi-agency developed and supported application to provide a next-generation mesoscale forecast model and data assimilation system to advance both the understanding and prediction of mesoscale weather and accelerate a transfer of research advances into operations. WRF has been parallelized using MPI to run on large-scale systems into the range of thousands of cores. It is in use by a number of agencies as well as by researchers. Due to its nature, WRF can produce a great deal of IO that can quickly grow with the model size.

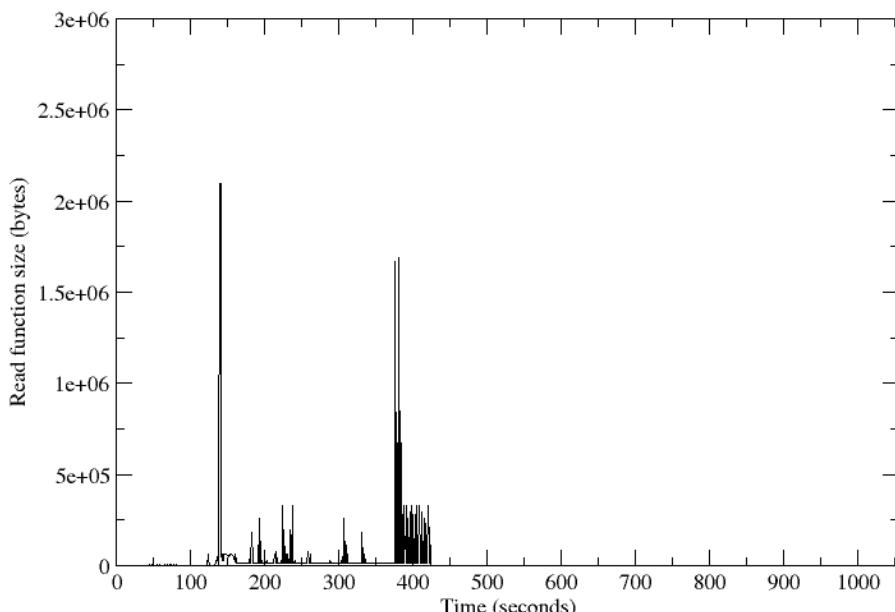


Fig. 1. Read Function size (amount of data) per call as a function of time during the execution of the application (referenced to the beginning of the execution)

There are several test cases available for WRF. This paper used the *conus12km_data_v3 dataset*. It was run on a test cluster at the University of Texas Advanced Compute Center (TACC). It was run on 16 cores of Intel® Xeon™ 5450 (code-named Harpertown) processors that have quad-cores per socket. The processors ran at 3.0 GHz and have 16KB L1 cache (data), and 6MB L2 cache with a 1333MHz FSB (Front-Side Bus). The nodes were Dell™ M600 blades in a Dell M1000e blade chassis. A total of two dual-socket, quad-core nodes were used and they were connected using DDR InfiniBand and the OFED 1.xx stack. The HCA's are Mellanox® DDR InfiniBand mezzanine cards and the switch is a Cisco Topspin SFS7000 DDR InfiniBand switch.

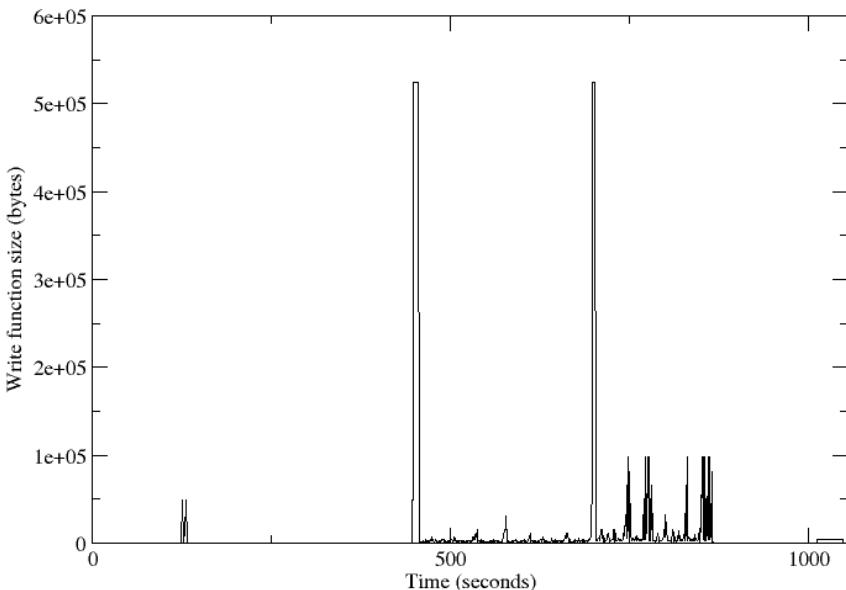


Fig. 2. Write Function size (amount of data) per call as a function of time during the execution of the application (referenced to the beginning of the execution)

The storage subsystem consisted of five 750GB 7,200 rpm SATA II hard drives that were used in a RAID-5 configuration on a Dell MD3000 that is connected to a host node via a Dell PERC5e RAID card. On the host node, the ext3 file system was used, and the storage was exported over the InfiniBand network using IPoIB (IP over InfiniBand) to all of the host nodes.

The nodes were running CentOS 4 with a 2.6.9-67.0.07.EL_lustre.1.6.5.1smp kernel that has added patches for Lustre support. The application was built with MVAPICH version 1.0.1a using the Intel 10.1 compilers.

5 Strace Sample Output

As an introduction to what IO pattern knowledge can be gleaned from strace output, some of the actual strace output from WRF is presented below. Each MPI process produced its own strace file, so each process can have its IO information tracked. The output snippet is from the rank 0 process.

```
13:31:12.151000 getcwd("/data/laytonjb/BENCHMARKS/WRF", 4096) = 30
13:31:12.151079 open("/data/laytonjb/BENCHMARKS/WRF/namelist.input",
O_RDWR) = -1 EACCES (Permission denied)
13:31:12.151322 open("/data/laytonjb/BENCHMARKS/WRF/namelist.input",
O_RDONLY) = 12
...
13:31:12.152176 lseek(13, 0, SEEK_CUR) = 0
```

The first three lines are the first three lines from the rank 0 process. The subsequent lines are taken from further in the strace output file. The first part of each line is the time the function began using the microsecond timing capability of strace. By computing the difference in times between two function calls, the elapsed time the function can be computed.

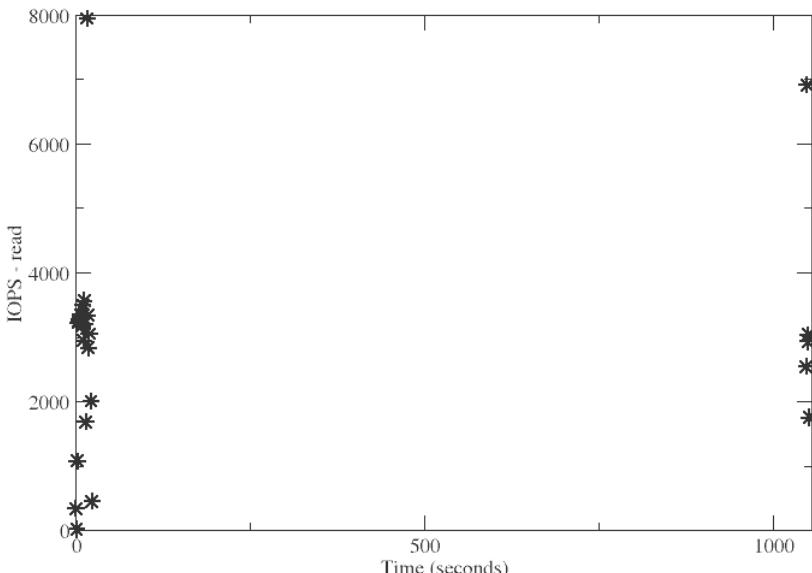


Fig. 3. Read IOPS (IO Operations Per Second) as a function of time during the execution of the application (referenced to the beginning of the execution)

However, just imagine doing that for a strace file that has over 100,000 lines and for possibly tens or hundreds of strace files (one for each MPI process). To assist with the processing and for gathering statistics, a tool named strace_analyzer, was written to process the strace files. Strace_analyzer is released under GPL 2.0. [3]

6 IO Patterns of WRF

Strace_analyzer was run against the strace output files from the previously discussed example (WRF). A total of 16 strace output files were processed (16 MPI processes). It was found that only the rank-0 process did any IO for the application itself. Therefore, the analysis presented will be only in regard to the rank-0 strace file. The total run time for the application was approximately 1200 seconds (20 minutes).

The first set of output from strace_analyzer is a simple table listing the major IO functions and the number of times they were called during the execution. The output snippet below shows that table with the IO command the number of times it was called during execution.

Number of Lines in strace file: 182575

-- IO Command Count --

Command	Count
<hr/>	
access	1
lseek	68807
fcntl	17
stat	38
unlink	3
open	453
close	354
fstat	67
read	75853
write	35744

After this output, the analyzer provides a quick summary of the write functions during the execution. It presents a breakdown of the size of the write functions as well as a summary.

-- File sizes for write() function --

IO Size Range (Bytes)	Number of Files
<hr/>	
(1) < 1K Bytes	35392
(2) 1K < < 8K Bytes	0
(3) 8K < < 32K Bytes	146
(4) 32K < < 1M Bytes	4
(5) 1M < < 1024G Bytes	0

-- WRITE SUMMARY --

Total number of Bytes written = 1,128,458,885 (1,128.458885 MB)

Number of Write function calls = 35,744

Average Bytes per call = 31,570.5820557296 (bytes) (0.0315705820557296 MB)

Time for slowest write function (secs) = 8.737661

Line location in file: 113329

Notice that the vast majority of the write functions used less than 1KB of data! There are only 4 write function calls that use up to 1MB of data and there are no function calls with more than 1MB of data.

The next section of the output contains the read function summary that mirrors the write summary. The vast majority of the read functions are less than 1KB, and there are no read functions greater than 1MB.

-- File sizes for read function() --

IO Size Range (Bytes)	Number of Files
-----------------------	-----------------

<hr/> <hr/>		
IO Size Range (Bytes)	Number of Files	
(1) < 1K Bytes	75823	
(2) 1K < < 8K Bytes	11	
(3) 8K < < 32K Bytes	4	
(4) 32K < < 1M Bytes	7	
(5) 1M < < 1024G Bytes	0	

-- READ SUMMARY --

Total number of Bytes read = 1,912,368,942 (1,912.368942 MB)

Number of Read function calls = 75

Average Bytes per call = 25,211.5136118545 (bytes) (0.0252115136118545 MB)

Time for slowest read function (secs) = 0.026381

Line location in file: 7652

The next section of the output presents the lseek function calls listed for each file descriptor. This section is important because of the impact of lseeks on reducing the throughput.

unit	Number of lseeks
------	------------------

<hr/> <hr/>	
unit	Number of lseeks
5	1
8	1
9	1
1	2
13	34386
12	34416

It is interesting to compare the number of seeks to the number of read and write functions. In this case, there is an lseek for about every read and/or write function.

The final section of the output lists the IOPS summary. For WRF, the maximum IOPS is fairly large for both the read and write functions. If one includes all IO functions, not just read and write, into a “Total” IOPS measurement, the IOPS is a very large number.

Maximum Write IOPS = 8014 occurred at 17 seconds

Average Write IOPS = 214

Maximum Read IOPS = 7940 occurred at 17 seconds

Average Read IOPS = 2917

Maximum Total IOPS = 31908 occurred at 17 seconds

Total IOPS = 1023

Final IOPS report

Write IOPS = 214

Read IOPS = 2917

Total IOPS = 1023

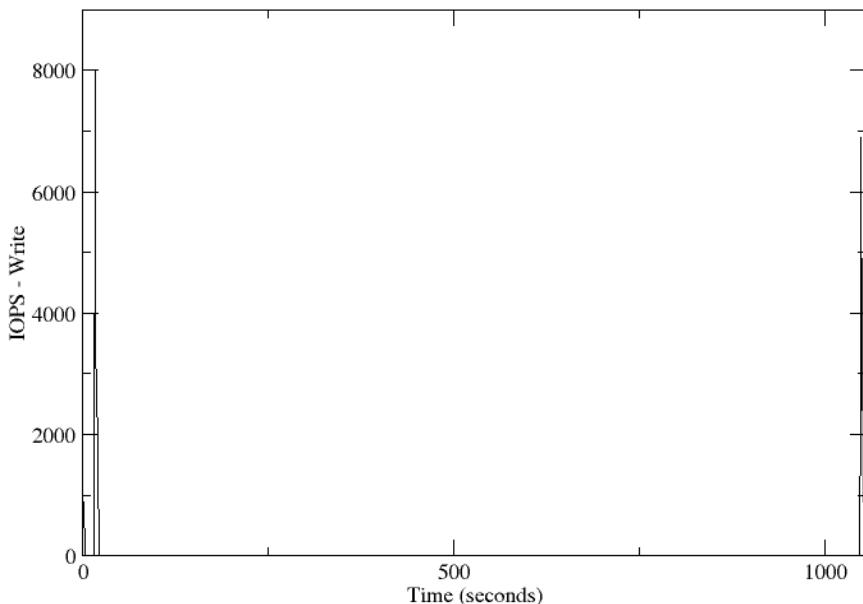


Fig. 4. Write IOPS (IO Operations Per Second) as a function of time during the execution of the application (referenced to the beginning of the execution)

The strace_analyzer also produces csv (comma separated value) files that can then be used in a spreadsheet for plotting or analysis. However, in some cases the output is too large for typical spreadsheets so the strace_analyzer will also produce simple data files that can be used in plot programs such as *grace*. [4]. Below are plots of the read and write history (number of bytes in read or write operation) of the execution referenced to the beginning of the execution. Along with these plots are the plots of the read and write IOPS.

7 Observations of IO Patterns of WRF

From the tables and plots, there are several interesting observations that can be made about the IO pattern of WRF for the data set tested:

- For the tested data set there was a total of approximately 1.1 GB of data written and a total of approximately 1.9 GB of data read.
- The vast majority of the write functions contain less than 1KB of data. But the average write function is for about 31.5KB.
- Determining whether the average write function is large or small depends upon your perspective. However, many HPC applications like to state that they use large sequential write functions. In many cases, this includes WRF. But the IO analysis shows that this isn't the case, and the average write size is small.
- For read functions, the vast majority of the functions are for less than 1KB. The average read function is for 25.2 KB.
- As for write functions, the read functions have a similar observation. Many HPC applications state that they do large sequential reads. However, for WRF, the average read is a fairly small at 25.2 KB.
- There is one lseek function for just about every read and write function. Lseek functions can potentially kill throughput performance so the fewer lseek functions used the better.
- Comparing the write/read function size plot vs. the corresponding IOPS plot, one can see that the peak IOPS occurs at the beginning and end of the execution. On the other hand the largest amount of write and read data is processed approximately in the middle of the run.

8 Summary

The observations of the IO patterns are interesting but some of it can be explained. WRF uses NetCDF for its output. [5] So the IO patterns are really attributable to NetCDF. Generally, NetCDF can be considered a database for scientific applications. Consequently, the fairly large number of IOPS is explained as NetCDF behaving somewhat like a database during small reads and writes while doing lseeks in between.

Based on the output and the observations, if one had to design an HPC Storage system for this application, it is likely to be something similar to that of a database. That is, the storage should be capable of a large number of IOPS and reasonable amounts of throughput. Given that the IOPS for the storage system tested are quite low, it

would be anticipated that adding a number of spindles, would produce a much larger IOPS capability and consequently greatly improve the performance. So the storage requirements that one would give prospective vendors would be something like the following:

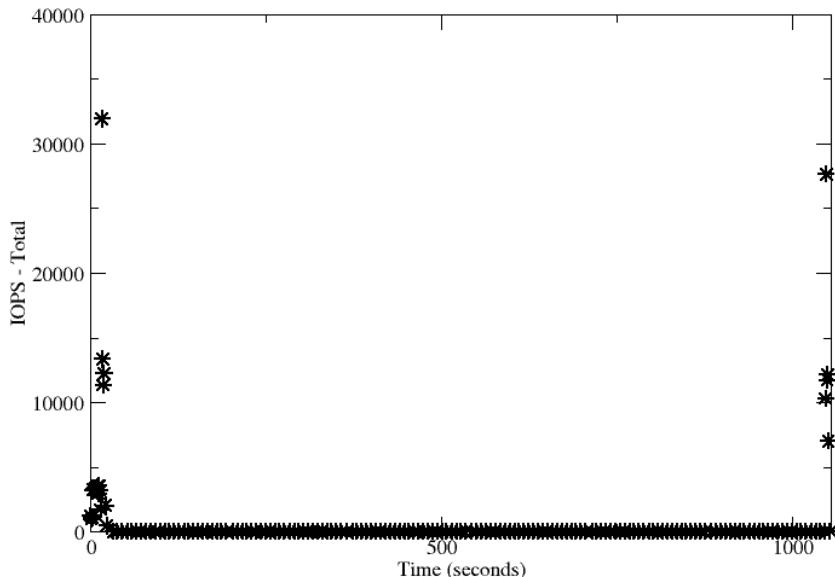


Fig. 5. Total IOPS (IO Operations Per Second) as a function of time during the execution of the application (referenced to the beginning of the execution). This includes all IO operations examined by the strace analyzer.

- The rank-0 process performs all of the IO for the application
- Application is IOPS driven (approximately 8,000 IOPS, read and write for nominal performance using NFS with IPoIB on DDR InfiniBand)
- Increased spindle count and/or high IOPS capable storage (e.g. SSD) are desired to improve IOPS
- Throughput is a secondary consideration although it should be reasonable (i.e. 500 MB/s as measured by iozone benchmark)
- It is expected that the application will be run on at least 16 nodes, expanding to 128 nodes as problem sizes grow over the life of the compute cluster.

This is a reasonably compact and workable set of specifications to help a vendor design an HPC Storage system. If one couples this type of statement with the stack ranked goals of the storage (e.g., (1) cost, (2) reliability, (3) ease of management, (4) ease of expandability, (5) performance, etc.) coupled with the process requirements, a vendor has a very good view of what you want to accomplish from the storage perspective and how it impacts application performance.

However, there is one aspect that has been left to the last but it perhaps the most important. That is, how much execution time was spent performing IO? This

fundamentally important question guides you in the development of the requirements. For WRF, which is known for doing a large amount of IO during the run, and for the configuration tested, the IO only consumed 2.17% of the execution time (22.87 seconds).

It would have been more logical to determine the amount of time spent doing IO at the beginning to avoid going through the process of examining the details of the IO pattern. However, in the author's experience almost no one can articulate how much run time is spent doing IO, even though it's perhaps the most important question. So, by putting this comment after the analysis, it emphasizes the point that this question should be answered first.

This brings up the question "is WRF IO bound?" The obvious answer is a pretty clear, "no" because the IO time is such a small percentage of the total execution time. Given this information would you then reconsider your set of requirements? It's fairly obvious that doubling the IO performance of the HPC Storage system will only improve the overall application performance by 1%. Recall that the test system used simple NFS over InfiniBand (IPoIB). Perhaps an alternative to improving the storage performance to improve application performance is to add more compute capability and to run on more compute nodes.

Having the ability to examine your application(s) for their IO patterns, and being able to make decisions and develop well thought out and defensible requirements makes the vendors job simpler, but more importantly, it makes your job easier because you understand what your HPC system is doing from an IO perspective.

Acknowledgments. The author wishes to thank Dr. Tommy Minyard at the University of Texas Advanced Computing Center (TACC) for his help in building and running WRF.

References

1. Skamarock, W.C., Klemp, J.B., Dudhia, J., Gill, D.O., Barker, D.M., Wang, W., Powers, J.G.: A description of the advanced research WRF version 2. Technical Report NCAR/TN-468+STR, National Center for Atmospheric Research (January 2007)
2. Dudhia, J., Gill, D., Henderson, T., Klemp, J., Skamarock, W., Wang, W.: The Weather Research and Forecast Model: Software Architecture and Performance. In: Zwiefelhofer, W., Mozdzynski, G. (eds.) Proceedings of the Eleventh ECMWF Workshop on the Use of High Performance Computing in Meteorology, pp. 156–168. World Scientific, Singapore (2005)
3. Cluster Buffer, strace_analyzer, <http://clusterbuffer.wetpaint.com>
4. Grace plotting tool, <http://plasma-gate.weizmann.ac.il/Grace/>
5. NetCDF (network Common Data Form),
6. <http://www.unidata.ucar.edu/software/netcdf/>
7. Running strace with MPI applications,
8. <http://www.delltechcenter.com/page/4-21-2008+-+strace+MPI+-+Comments>

Failure Data-Driven Selective Node-Level Duplication to Improve MTTF in High Performance Computing Systems

Nithin Nakka and Alok Choudhary

Department of Electrical Engineering and Computer Science
Northwestern University
2145 Sheridan Rd, Tech Inst. Bldg., EECS Dept., Evanston IL 60201
`{nakka, choudhar}@eecs.northwestern.edu`

Abstract. This paper presents our analysis of the failure behavior of large scale systems using the failure logs collected by Los Alamos National Laboratory on 22 of their computing clusters. We note that not all nodes show similar failure behavior in the systems. Our objective, therefore, was to arrive at an ordering of nodes to be incrementally (one by one) selected for duplication so as to achieve a target MTTF for the system after duplicating the least number of nodes. We arrived at a model for the fault coverage provided by duplicating each node and ordered the nodes according to coverage provided by each node. As compared to traditional approach of randomly choosing nodes for duplication, our model - driven approach provides improvements ranging from 82% to 1700% depending on the improvement in MTTF that is targeted and the failure distribution of the nodes in the system.

Keywords: Fault-tolerance, Mean Time To Failure, Duplication, Partial Duplication.

1 Introduction

Computers are being employed increasingly in highly mission- and life-critical and long-running applications. In this scenario, there is a corresponding demand for high reliability and availability of the systems. Since failures are inevitable in a system, the best use of the bad bargain is to employ fault-detection and recovery techniques to meet the requirements. Mean Time to Failure (MTTF) (commonly also referred to as Mean Time Between Failures (MTBF)) is an important well-accepted measure for the reliability of a system. MTTF is the time elapsed, on an average, between any two failures in the component being studied.

Broadly speaking, two types of applications demand high reliability – (i) those which can be stopped and their state captured at a suitable point and their execution resumed at a later point in time from the captured state, also called the checkpointed state, (ii) those programs that cannot be interrupted and need to execute for a minimum amount of time, till the application (or the mission) is completed. Most long-running scientific applications are examples of applications in the former category.

They require efficient mechanisms to take checkpoints of the entire state of the application at a suitable point, and effective techniques to detect failures so as to roll back execution to the checkpointed state. For applications in the latter category, such as systems and software for flight, or spacecraft control, a time for the length of the mission (or mission time) is pre-determined and appropriate fault-tolerance techniques need to be deployed to ensure that the entire system does not fail within the mission time. The mission time of a flight system directly determines the length of its travel and is a highly critical decision point.

The MTTF of a system is an estimate of the time for which the system can be expected to work without any failures. Therefore, for applications that can be checkpointed MTTF could be used to determine the checkpointing interval, within which the application's state must be checkpointed. This would ensure that the checkpoint state itself is not corrupted and hence by rolling back to this state on detecting a failure the application will continue correct execution. For applications of the latter category, the MTTF can be used to determine the mission time, before which the system executes without any failure.

Understanding the failure behavior of a system can greatly benefit the design of fault-tolerance and reliability techniques for that as well as other systems with similar characteristics and thereby increasing their MTTF. Failure and repair logs are a valuable source of field failure information. The extent to which the logs aid in reliable design depends on the granularity at which the logging is performed. System level logs could assist in system-wide techniques such as global synchronous checkpointing etc. However, logging at a finer granularity, like that at the node-level, improves the effectiveness of techniques applied at the node level. An important observation that we make in this paper is that, "All nodes in a system are not equal" (either by functionality or by failure behavior). We attempt to take into account the logical (in the network configuration) locations of a node in the system, in reconciling its observed failure behavior.

Node-level reliability information also helps in designing and deploying techniques such as duplication selectively to the most critical portions of the system. This decreases setup, maintenance and performance costs for the system. Furthermore, with a framework for selective fault-tolerance in place the techniques could be customized to meet the specific reliability requirements of the application. In this paper we show how node-level duplication can be customized to meet an application's target MTTF.

The key contributions of this work are:

1. Analysis of node-level failures and their correlation with system and network configuration.
2. Data-driven estimation of the coverage provided for selective duplication of nodes in the system.
3. A methodology for selecting an optimal or near-optimal subset of the nodes that need to be duplicated to achieve the MTTF requirements of the application.

Description of Systems under Study and Data Sets

Los Alamos National Laboratory has collected and published data regarding the failure and usage of 22 of their supercomputing clusters. This data was previously analyzed by Schroeder et. al. from CMU to study the statistics of the data in terms of the root cause of the failures, mean time between failures and the mean time to repair.

Table 1 provides the details of the systems. In the “Production Time” column the range of times specifies the node installation to decommissioning times. A value of “N/A” indicates that the nodes in the system have been installed before the observation period began. For these machines, we consider the beginning of the observation period (November 1996) as the starting date for calculating the production time. For some machines, multiple ranges of production times have been provided. This is because the systems have been upgraded during the observation period and each production time range corresponds to set of nodes within the system. For the sake of simplicity, for all such machines, we consider the production time to be the longest range among the specified ranges.

The failure data for a single system includes the following fields:

System: The number of the system under study (as referred to by Los Alamos Laboratory)

machine type: The type of the system (smp, cluster, numa)

number of nodes: The number of nodes in the system

number of processors per node: Some systems are heterogenous in that they contain nodes with different number of processors per node

number of processors total: Total number of processors in the system

$$\sum_{i=0}^{k} n_i \cdot p_i , \text{ where for a system, } k \text{ is the number of node types in the system, } n_i \text{ is the number of nodes of type } i \text{ and } p_i \text{ is the number of processors in a node of type } i .$$

node number starting with zero: This node numbering is provided by Los Alamos so as to maintain consistency among all systems in terms of node numbering for easier comparison of systems.

install date: Date that the node was installed in the system. Since the systems under study were upgraded during the period of study, this field can vary for nodes within a system

production date: Date, after the installation date, where the node has been tested for initial burn-in effects and known operational failures, so that the node could be used to run production applications.

decommission date: Date that the node was removed from the system.

field replacable unit type: Whether on a failure the repair procedure involves replacing the entire unit or a portion of it.

memory: the amount of memory on the node

node purpose: The function of the node in the overall system: (i) compute, (ii) front end, (iii) graphics or a combination of the three such as graphics.fe (both graphics and front end).

Problem Started (mm/dd/yy hh:mm): The time of occurrence of the failure event. This is logged by an automated fault detection engine.

Problem Fixed (mm/dd/yy hh:mm): The time at which the failure was repaired and the system restored. This is logged by the administrators and repair staff.

Down Time: The time elapsed between the occurrence of the problem and restoration of the system.

The root cause for the failures is classified in the following categories: (i) Facilities, (ii) Hardware, (iii) Human Error, (iv) Network, (v) Undetermined, or, (vi) Software.

Table 1. Characteristics of Systems under study

System		System Number	Num nodes	# procs in node	# procs	Network Topology	Production Time	Remarks
Type	Category							
A	smp	7	1	8	8	Not Applicable	N/A – 12/99	
B	smp	24	1	32	32	Not Applicable	N/A – 12/03	
C	smp	22	1	4	4	Not Applicable	N/A – 04/03	
D	cluster	8	164	2	328	GigaBit Ethernet Tree	04/01 – 11/05 u12/02 – 11/05	
E	cluster	20	256	4	1024	Dual rail fat tree	12/01 – 11/05	
	cluster	21	128	4	512		09/01 – 01/02	
	cluster	18	1024	4	4096		05/02 – 11/05	
	cluster	19	1024	4	4096		10/02 – 11/05	
	cluster	3	128	4	512	Single rail fat tree	09/03 – 11/05	
	cluster	4	128	4	512		09/03 – 11/05	
	cluster	5	128	4	512		09/03 – 11/05	
	cluster	6	32	4	128		09/03 – 11/05	
F	cluster	14	128	2	256	Single rail fat tree	09/03 – 11/05	Most jobs run on only one 256-node segment
	cluster	9	256	2	512		09/03 – 11/05	
	cluster	10	256	2	512		09/03 – 11/05	
	cluster	11	256	2	512		09/03 – 11/05	
	cluster	13	256	2	512		09/03 – 11/05	
	cluster	12	512	2	1024		09/03 – 11/05 03/05 – 06/05	
G	cluster	16	16	128	2048	Multi rail fat tree	12/96 – 09/02	
	cluster	2	49	128, 80	6152		01/97 – 11/05	
	cluster	23	5	128, 32	544		06/05 – 11/05	
H	numa	15	1	256	256		10/98 – 12/04	
							01/98 – 12/04	
							11/02 – 11/05	
							11/05 – 12/04	
							11/04 – 11/05	

2 Related Work

There has been significant study over the past few decades on analyzing failure logs from large-scale computers to understand the failure behavior of such systems and possibly use the knowledge in improving system design. Plank et. al. [0] derive an optimal checkpointing interval for an application using the failures logs obtained from networks of workstations. They derive the failure rate of the machines using the time between failures observed in the logs. In another study, Nath et. al. [0] study real-world

failure traces to recommend design principles that can be used to tolerate correlated failures. Particularly they have used it in recommending data placement strategies for correlated failures.

Prior work in analyzing failure rates tried to arrive at reasonable curves that fit the failure distribution of the systems 0-0. Schroeder and Gibson 0 have presented the characteristics of the failure data that has been used in this study as well. However, a limitation of these studies is that they do not evaluate how system design choices could be affected by the failure characteristics that they have derived from the data. In this work we specific attempt to understand the failure behavior and use that knowledge in the design of an optimal node-level duplication strategy with the aim of increasing the overall availability of the system at the least cost.

There is also interesting research work in understanding the correlations between system parameters and failure rate. Sahoo et. al. 0 show that the workload of the system is closely correlated with its failure rate, whereas Iyer 0 and Castillo 0 bring out the correlation between workload intensity and the failure rate. In our study we study the dependency of failure rate on network configuration, which in turn determines the workload characteristics of the system. For example, a fat tree topology necessitates higher communication and computation bandwidth and load at the higher levels of the tree structure.

Oliner and Stearley 0 have analyzed system logs from five supercomputers and critically evaluate the interpretations of system administrators from patterns observed in the logs. They propose a filtering algorithm to identify alerts from system logs. They also recommend enhancements to the logging procedures so as to include information crucial in identifying alerts from non-alerts.

Lan et. al. 00 have proposed a machine-learning based automatic diagnosis and prognosis engine for failures through their analysis of the logs on Blue Gene/L systems deployed at multiple locations. Their goal is to feed the knowledge inferred from the logs to checkpointing and migration tools 00 to reduce the overall application completion time.

Oliner et. al. 0 derive failure distributions from multiple supercomputing systems and propose novel job-scheduling algorithms that take into account the occurrence of failures in the system. They evaluated the impact of this on the average bounded slowdown, average response time and system utilization. In our study we have utilized failure and repair time distributions to propose a novel node-level duplication strategy. The metric of evaluation used is the mean time to failure (MTTF) of the system.

Duplication, both at the system- and node- level has been a topic of active and extensive research in the micro-architecture and fault -tolerant computing areas. *Error Detection Using Duplicated Instructions (EDDI)* 0 duplicates original instructions in the program but with different registers and variables. Duplication at the application level increases the code size of the application in memory. More importantly, it reduces the instruction supply bandwidth from the memory to the processor. *Error Detection by Diverse Data and Duplicated Instructions (ED⁴I)* 0 is a software-implemented hardware fault tolerance technique in which two “different” programs with the same functionality are executed, but with different data sets, and their outputs are compared. The “different” programs are generated by multiplying all variables and constants in the original program by a diversity factor k .

In the realm of commercial processors the IBM G5 processor 0 has extra I- and E-units to provide duplicate execution of instructions. To support duplicate execution, the G5 is restricted to a single-issue processor and incurs 35% hardware overhead.

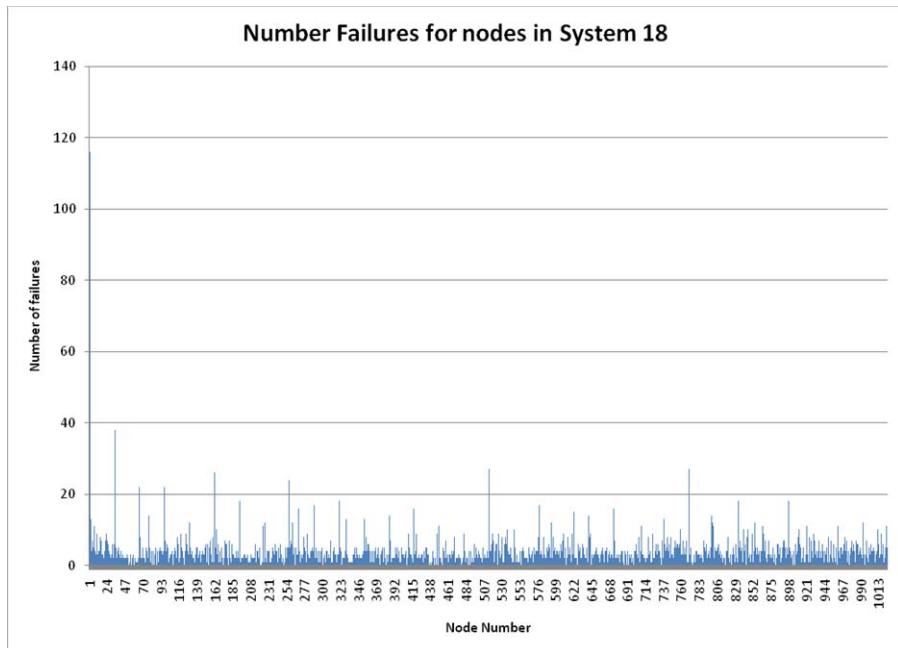
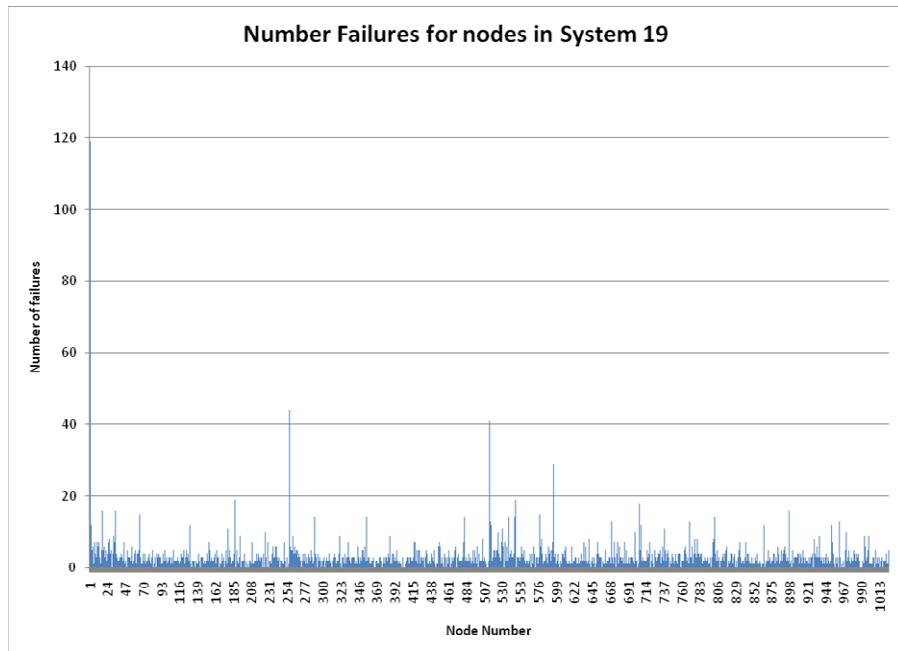
In experimental research, simultaneous multithreading (SMT) 0 and the chip multiprocessor (CMP) architectures have been ideal bases for space and time redundant fault-tolerant designs because of their inherent redundancy. In simultaneously and redundantly threaded (SRT) processor, only instructions whose side effects are visible beyond the boundaries of the processor core are checked 0-0. This was subsequently extended in SRTR to include recovery 0. Another fault-tolerant architecture is proposed in the DIVA design 00. DIVA comprises an aggressive out-of-order superscalar processor along with a simple in-order checker processor. *Microprocessor-based introspection (MBI)* 0 achieves time redundancy by scheduling the redundant execution of a program during idle cycles in which a long-latency cache miss is being serviced. SRTR 0 and MBI 0 have reported up to 30% performance overhead, which are close to the overhead observed for Full Duplication (FullRep) in our experiments. These results counter the widely-used belief that full duplication at the processor-level incurs little or no performance overhead.

SLICK 0 is an SRT-based approach to provide partial replication of an application. The goals of this approach are similar to ours. However, unlike this approach we do not rely on a multi-threaded architecture for the replication. Instead, this paper presents modifications to a general superscalar processor to support partial or selective replication of the application.

As for research and production systems employing system-level duplication, the space mission to land on the moon used a TMR enhanced computer system 0. The TANDEM, now HP, Integrity S2 computer system 0 provided reliability through the concept of full duplication at the hardware level. The AT&T No.5 ESS telecommunications switch 0, 0 uses duplication in its administrative module consisting of the 3B20S processor, an I/O processor, and an automatic message accounting unit, to provide high reliability and availability. The JPL STAR computer 0 system for space applications primarily used hardware subsystem fault-tolerant techniques, such as functional unit redundancy, voting, power-spare switching, coding, and self-checks.

3 Failure Rate Correlation with Network Configuration

The failure data for each system is analyzed independently. The data set contains the data for all the systems in a monolithic fashion. Therefore, the data is split based on the system number. For data for a specific system, using the nodenumz field, which gives the number of the failed node, the failures for each node are aggregated and the number of failures for each node calculated. This data provides an ordering of the nodes based on the number of failures experienced by the node within the observation period. The number of failures per node also provides insights into the correlation between the node's failure behavior and the logical location of the node in the system's network configuration.

**Fig. 1.** Number of failures for nodes in System 18**Fig. 2.** Number of failures for nodes in System 19

As mentioned before, the nodes of a particular system show variability in the failure rate. This could partly be explained by the network configuration of the distributed system.

Fig. 1 and Fig. 2 show the failure behavior of individual nodes in System 18 and 19. The figure shows that the peaks of failures are at nodes with numbers that are multiples of 32, with a few exceptions. The exception in the failures of other nodes could be due to their specific workload characteristics or due to the inherent failure susceptibility of that node. On discussion with the system administrators, it was revealed that these systems are configured such that every 32nd node serves as an NFS root for the other 31 nodes in the group and is running NFS root type services, making its workload considerably higher than the rest.

4 Approach

This section describes our approach for analyzing the data and building a model used for selective node-level duplication. The records for a single node are ordered according to the time of occurrence of the failure, as given by the “Prob Started field”. The time elapsed between the start of one failure and the consecutive failure in this ordered list gives the time between these two failures. Following this procedure the times between failure for each pair of failures for the node are calculated. The average of all these times is used as an estimate for the mean time between failures or the mean time to failure (MTTF) for the node. Time To Failure (TTF) for a fault $i = TTF_i = (prob\ started)_i - (prob\ started)_{i-1}$. Therefore,

$$MTTF = \frac{\sum_i^n TTF_i}{n} = \frac{(prob\ started)_n - (system\ install\ date)}{n} \text{ OR}$$

MTTF is calculated as:

$$MTTF = \frac{\text{Total Period of study}}{\text{Number of Failures observed (n)}} \quad \text{Eq. 1}$$

The period of study of a system is its production time, defined elsewhere as the time between its installation and its decommissioning or the end of the observation period, whichever occurs first.

The “Downtime” field provides the time required by the administrators to fix the failure and bring the system back to its original state. It can be calculated as the difference between the “Prob Ended” and “Prob Started” fields. This is the repair time for this failure. Averaging this field over all the records for a single node provides an estimate for the mean time to repair (MTTR) for this node. Time To Repair (TTR) for a failure $i = TTR_i = (prob\ fixed)_i - (prob\ started)_i$. Therefore, Mean Time To Repair is given by

$$MTTR = \frac{\sum_i^n TTR_i}{n}$$

4.1 Introducing Node-Level Duplication

From the previous analysis procedures, the MTTF and the MTTR for a node have been estimated. Now, we introduce a methodology to understand the effect on node failure if we augment it with a spare node. Fig. 3 shows the states through which a duplicated node transitions on failure and repair events. When both the original and the spare node are working correctly the system is in state “2”. It is assumed that, after a failure is detected in the node, the system has the reconfiguration capability to fail over to the spare node instantaneously. Computation therefore continues uninterruptedly. This state of the node is represented by state “1” in the figure. In the mean time, the original node is repaired. The roles of the spare node and original node are switched. If no other failure occurs in the node, before the original node is repaired, then the original node assumes the role of the spare node, while the computation continues on the spare node. Essentially, the system is brought back to its pristine, fault-free state (State “2” in the figure). However, if the next failure for the node occurs within the mean time to repair for that node, then it is not possible to continue computation on that node. The node reaches state “0”. We declare that duplication cannot cover this second failure. There are other possible transitions between these states, shown as dotted lines in Fig. 3. They are (i) State “0” to “2”: When both the original and spare nodes are repaired and the node returns to its normal state. (ii) State “0” to “1”: When one of the failed nodes (the original or the spare) is repaired and computation continues on this node. (iii) State “2” to “0”: When both nodes fail at the same time. However, it is to be noted that for the analysis based on the data these transitions need not be considered. There would not be a transition from State “2” to “0” since the data represent failures only in one single node and would not therefore have

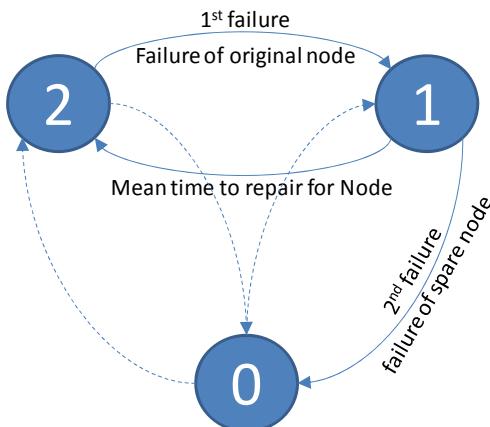


Fig. 3. State transition diagram for node failure with duplication

two simultaneous failures. The purpose of the analysis (aided by the state transition diagram) is to decide whether a particular failure can be covered by duplication or not. Once the node reaches State “0” it is declared that the failure cannot be covered by duplication. Therefore, outward transitions from State “0” (to States “1” and “2”) are not considered.

Based on this analysis, conducted for each node individually, we evaluate all the failures that are covered by duplication. This analysis provides an estimate of the nodes that benefit the most (have the maximum reduction in the number of failures) when they are duplicated.

The next part of the study is used to achieve application requirements of MTTF. To estimate the improvement in MTTF using node-level duplication, the nodes are ordered in terms of the reduction in number of failures when they are duplicated. In short, we will refer to this sorting criterion as the duplication coverage of the node. For a system without duplication the MTTF is calculated as the ratio between the total period of observation for the system and the number of failures observed as in the original data. Now, we pick the node with the highest duplication coverage. The MTTF for the system is recalculated as the total observation period for the system divided by the remaining number of failures in the system. Since the total observation period is constant, the MTTF is inversely proportional to the number of failures remaining in the system. Next, we pick the node with the next highest duplication coverage and recalculate the MTTF. Progressively, we include nodes in the order of their duplication coverage and calculate the MTTF. The duplication of nodes is continued until the MTTF requirements of the application are met.

4.2 Effect of Duplication on System Reliability

The node-level failure information is interpolated by selectively applying duplication to nodes that are most sensitive to errors. Depending on its execution profile, or on the frequency of its checkpointing support, the application demands a certain mean time to failure from the system so as to reduce the amount of wasted computation in the event of a failure. We study how the MTTF of the system is affected by incrementally adding dual redundancy to the nodes in the system.

In this study we consider node-level duplication as a fault-tolerance technique employed to increase the MTTF of a system. For each node that is duplicated, using the analysis mentioned in Section 0, some or all of the failures for that node are covered. This decreases the total number of failures in the system, thereby increasing the MTTF of the system using Eq. 1. If no information about the failure rate of nodes is available, all nodes would be assumed to fail uniformly and therefore the nodes to be duplicated would be picked at random one after another. We refer to this as the Traditional approach. However, with the failure rate analysis of the data we have derived a model for the coverage provided by duplication at the node-level and an ordering for the nodes based on the improvement in coverage by duplicating that node. Using this model, we may be able to improve the choice of nodes to be duplicated. This is the “Model-driven” approach to selecting nodes for incremental duplication.

Fig. 4 through Fig. 9 compare the cumulative increase in MTTF due to incremental node duplication for Systems 10, 11, 12, 13, 18, and 19 respectively, for these two approaches. The x-axis represents the percentage of nodes that have been duplicated.

The y-axis gives the percentage in MTTF achieved by duplicating that subset of the nodes in the system. The data points when major increments in MTTF (25%, 50%, 100%, 150%, 200%)¹ are crossed are labeled on the curves for the “Traditional” and “Model-driven” approaches. We see that consistently for all curves and for all major increments in MTTF, our Model-driven approach far outperforms the Traditional approach of random selection of nodes. For example, in Fig. 4 for System 10 the model-driven approach achieves close to 25% improvement of MTTF (exactly 26.7%) when 3.9% of the nodes are duplicated. For approximately the same improvement in MTTF the traditional approach requires 20.3% of the nodes to be duplicated. Thus there is an improvement of about 420% in the number of nodes to be duplicated using our data-derived model-driven approach. Table 2 summarizes these results for 6 systems that characterize those under study and for different aims for improvement in MTTF.

From Table 2 we note two striking observations. Firstly, the average improvement in number of nodes duplicated varies widely between different systems. This is due to the variation in the failure distribution of the system. The marginal increase in fault coverage for System 18, System 10 and System 12 with every additional node duplicated using the model-driven approach is shown in Fig. 10, Fig. 11, and Fig. 12 respectively. We see that for Systems 18 and 19, using the model-driven approach,

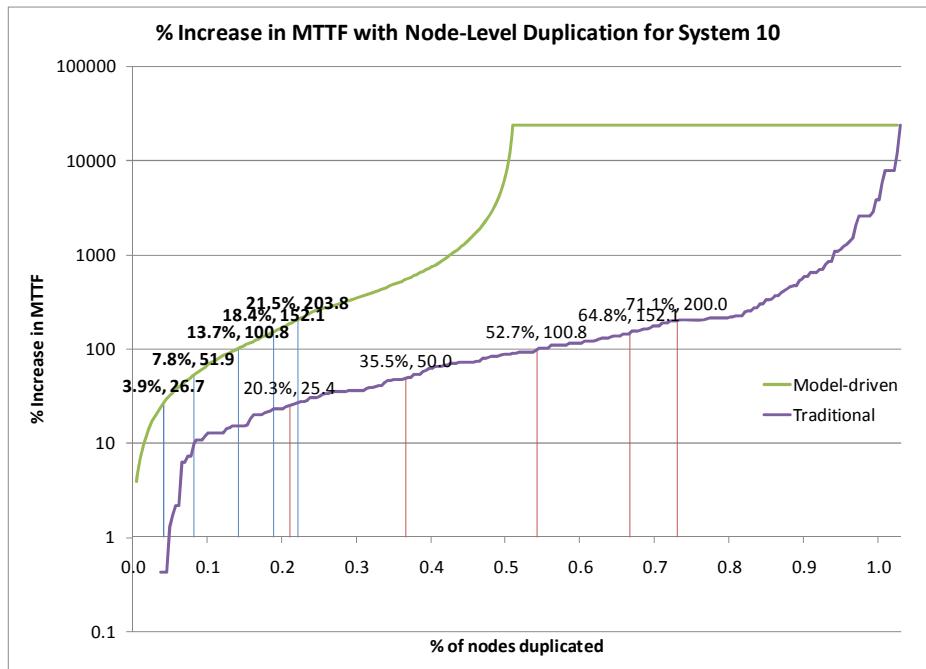


Fig. 4. % Increase in MTTF with node-level duplication for System 10

¹ Note that the actual data point represents the first time the increase in MTTF has *crossed* a major increment and is not exactly equal to the major increment.

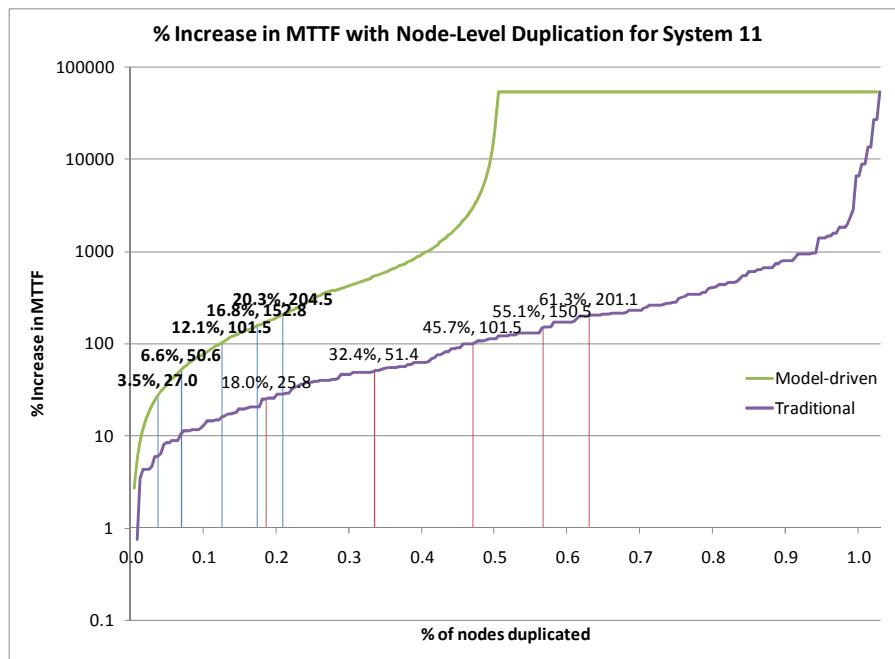


Fig. 5. % Increase in MTTF with node-level duplication for System 11

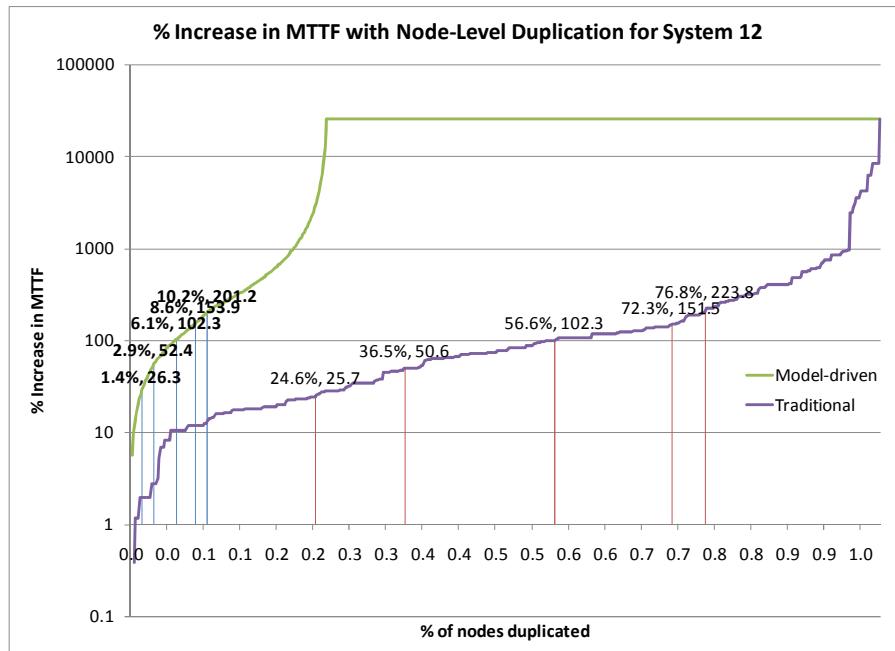


Fig. 6. % Increase in MTTF with node-level duplication for System 12

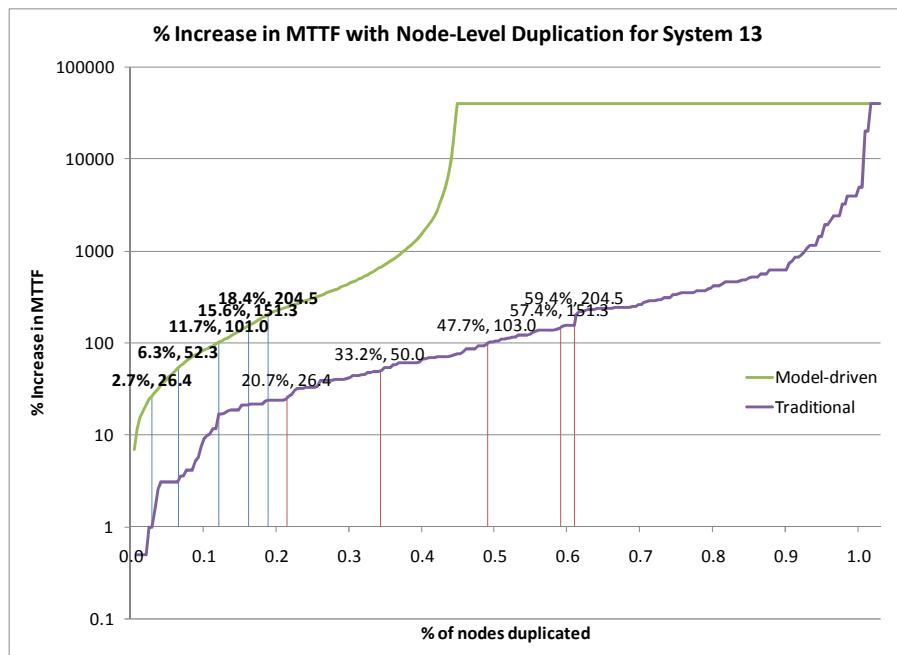


Fig. 7. % Increase in MTTF with node-level duplication for System 13

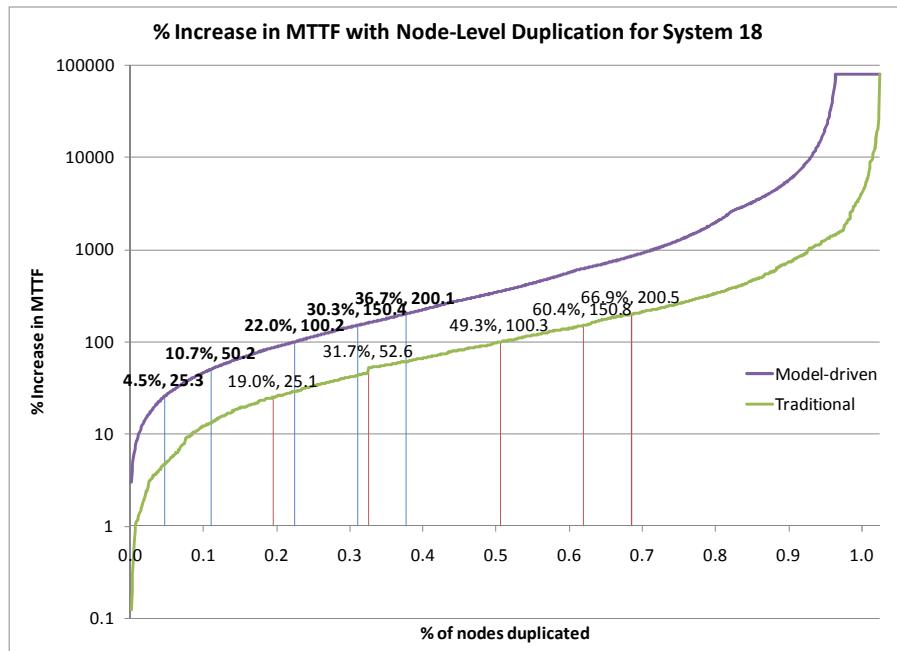


Fig. 8. % Increase in MTTF with node-level duplication for System 18

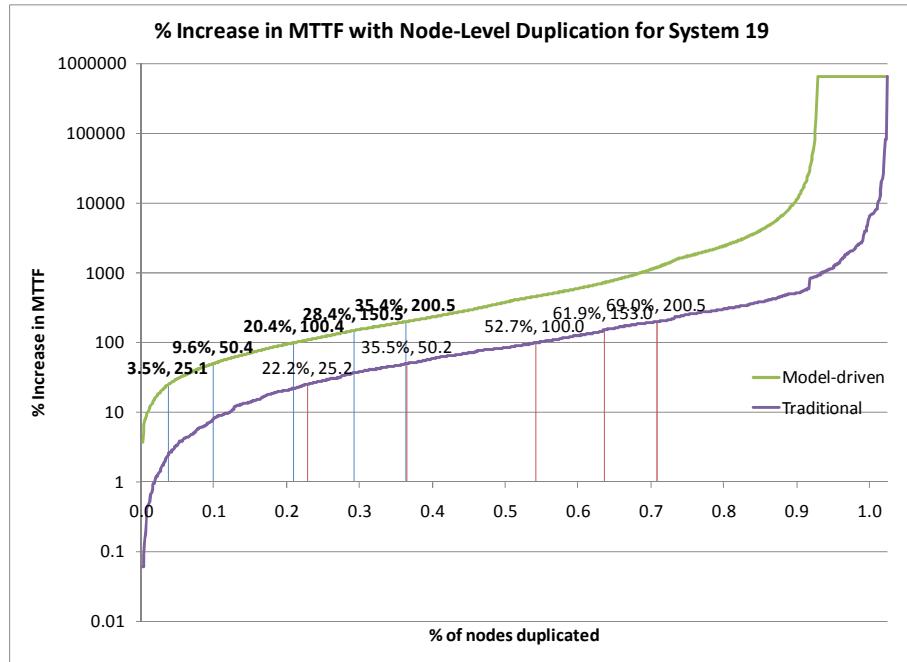


Fig. 9. % Increase in MTTF with node-level duplication for System 19

Table 2. Improvement in #nodes duplicated using Model-driven vs Traditional approaches

Improvement in nodes duplicated		System Number						Average
		10	11	12	13	18	19	
%MTTF Improvement (approx.)	25	420 %	411%	1700%	657%	324 %	531%	674%
	50	355 %	388%	1147%	431%	195 %	271%	465%
	100	286 %	277%	835%	307%	124 %	158%	331%
	150	253 %	228%	741%	268%	100 %	118%	285%
	200	231 %	202%	656%	223%	82%	95%	248%
Average		309 %	301%	1016%	377%	165 %	235%	

the MTTF continues to increase until about 90% of the nodes are duplicated. This indicates that failures occur in upto 90% of the nodes, even though to a small degree, as shown in Fig. 10. Therefore the average improvement for these systems is 165%

and 235% respectively. For Systems 10, 11 and 13, it is seen that failures occur in only about 50% of the nodes (please see Fig. 11), thus there is an initial rapid improvement in MTTF using the model-driven approach as compared to the traditional approach. The average improvement for these systems ranges between 300 to 375%. For System 12, failures are concentrated in only upto 25% of the nodes, as shown in Fig. 12. Therefore, the improvement is even more stark and rapid, thereby showing an average improvement of about 1000%.

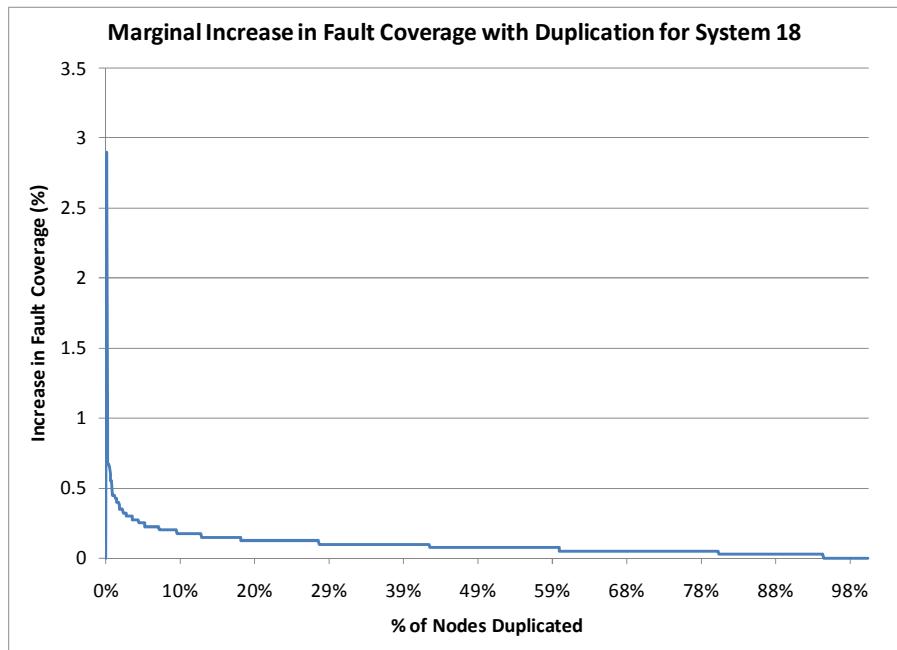


Fig. 10. Marginal increase in fault coverage with model-driven node duplication for System 18

Secondly, the improvement achieved in the nodes to be duplicated decreases as we try to achieve higher increments in MTTF. In other words, the gain in number of nodes duplicated for 25% improvement in MTTF is higher than that for 50% improvement in MTTF, which in turn is greater than that for a 100% improvement in MTTF and so on. This supports intuition since the model-driven approach is essentially greedy. It attempts to pick the node that provides the next best improvement in MTTF (or failure coverage). Therefore, the marginal increase in failure coverage for every additional node that is duplicated decreases monotonically. However, for a random selection of nodes, it is possible that nodes selected at later stages for duplication have a higher increment in the failure coverage than that of nodes selected in the earlier picks. These two trends are depicted in Fig. 13.

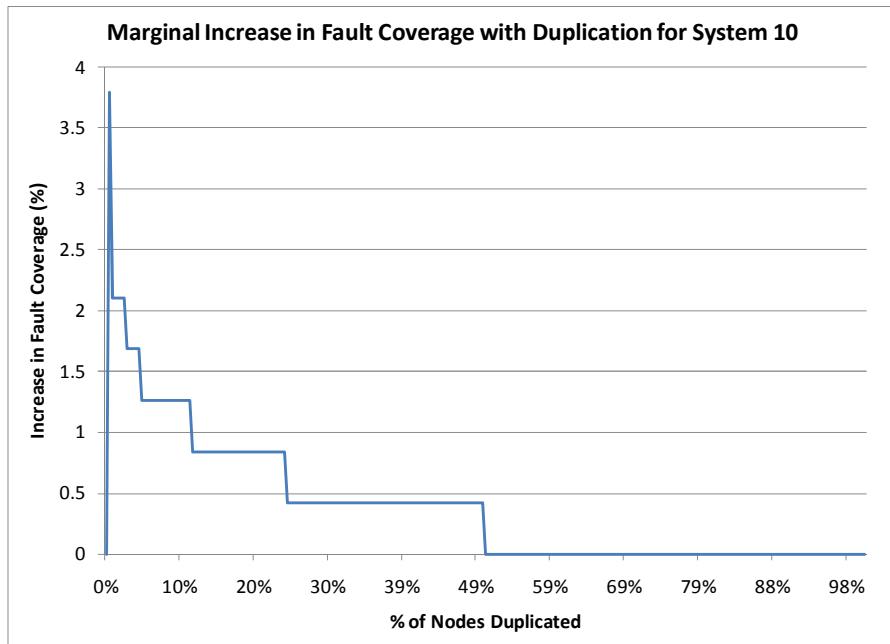


Fig. 11. Marginal increase in fault coverage with model-driven node duplication for System 10

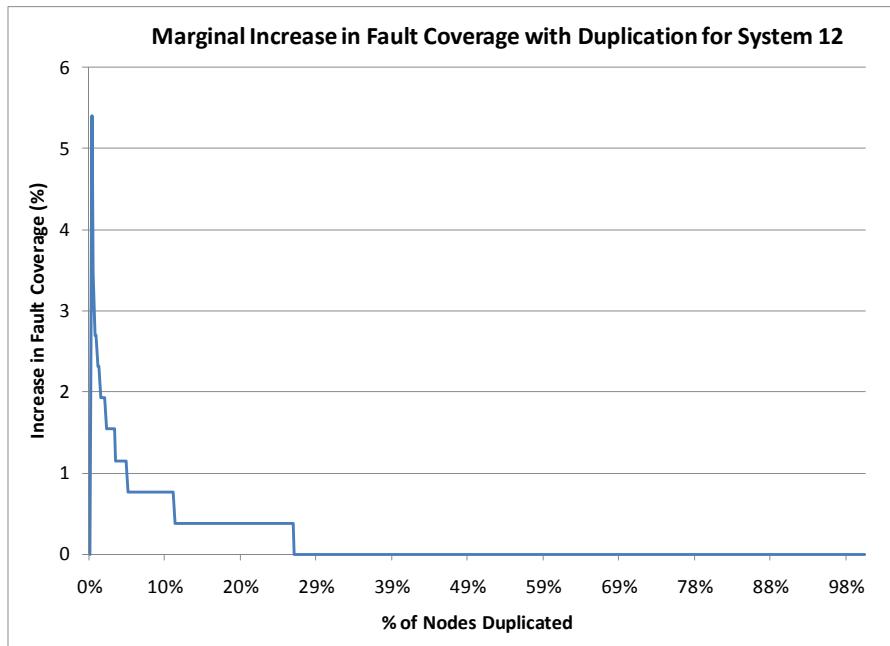


Fig. 12. Marginal increase in fault coverage with model-driven node duplication for System 12

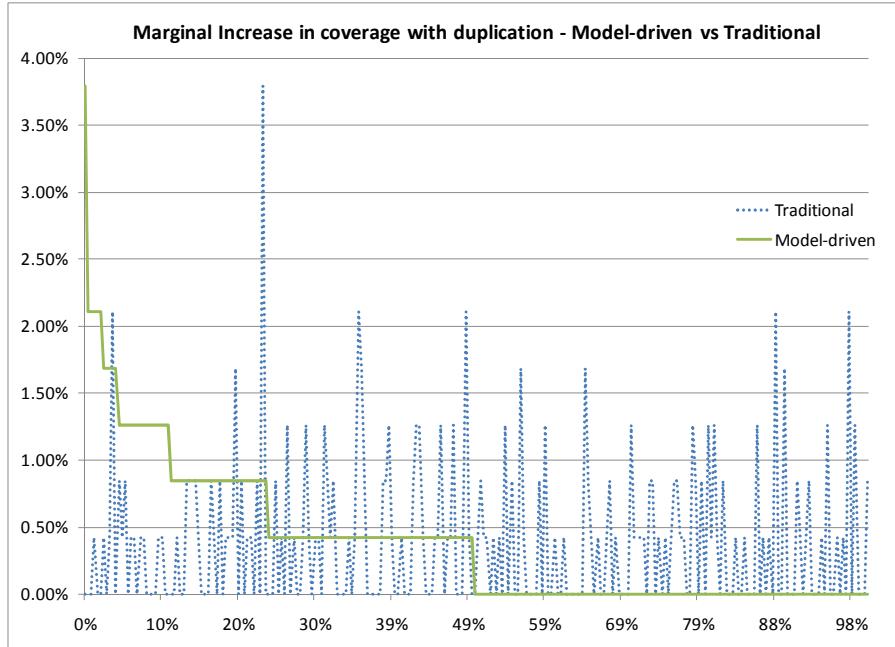


Fig. 13. Marginal increase in failure coverage for traditional and model-driven approaches

5 Conclusions and Future Directions

In this paper, we have presented our analysis of the failure behavior of large scale systems using the failure logs collected by Los Alamos National Laboratory on 22 of their computing clusters. We note that not all nodes show similar failure behavior in the systems. Our objective, therefore, was to arrive at an ordering of nodes to be incrementally (one by one) selected for duplication so as to achieve a target MTTF for the system after duplicating the least number of nodes. The network configuration of the systems and the consequent unequal distribution of workloads showed a close correlation with the failure behavior of the nodes. Using the start times and the down times logged for the failures we derived the time between failures and the mean time for repairs failures on a node. Using these two derived quantities, we arrived at a model for the fault coverage provided by duplicated each node and ordered the nodes according to coverage provided by each node. We saw that as compared to traditional approach of randomly choosing nodes for duplication, our model-driven approach provides improvements ranging from 82% to 1700% depending on the improvement in MTTF that is targeted and the failure distribution of the nodes in the system. The variations due to these parameters have also been explained.

The failure data from LANL provides node level failure information even though each node has multiple processors. Also, different nodes have different number of processors, therefore a more fine-grained logging of failures at the processor-level could provide even higher improvement in hardware overheads in achieving higher levels of System-level MTTFs.

Acknowledgments. This work was supported in part by NSF OCI-0956311, NSF HECURA CCF-0621443, NSF SDCI OCI-0724599, NSF CNS-0830927, NSF IIS-0905205, and DOE SCIDAC-2: Scientific Data Management Center for Enabling Technologies (CET) grant DE-FC02-07ER25808, DOE FASTOS award number DE-FG02-08ER25848. We are very thankful to Gary Grider, John Bent, John Nunez and Satsangat Khalsa from Los Alamos National Laboratory (LANL) for their guidance and providing key insights into the data.

References

1. Plank, J.S., Elwasif, W.R.: Experimental assessment of workstation failures and their impact on checkpointing systems. In: Proceedings of Fault Tolerant Computing Systems, FTCS 1998 (1998)
2. Nath, S., Yu, H., Gibbons, P.B., Seshan, S.: Subtleties in tolerating correlated failures. In: Proceedings of the Symposium On Networked Systems Design and Implementation, NSDI 2006 (2006)
3. Heath, T., Martin, R.P., Nguyen, T.D.: Improving cluster availability using workstation validation. In: Proceedings of ACM SIGMETRICS (2002)
4. Long, D., Muir, A., Golding, R.: A longitudinal survey of internet host reliability. In: Proceedings of the 14th Intl. Symposium on Reliable Distributed Systems (1995)
5. Nurmi, D., Brevik, J., Wolski, R.: Modeling machine availability in enterprise and wide-area distributed computing environments. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 432–441. Springer, Heidelberg (2005)
6. Sahoo, R.K., Sivasubramaniam, A., Squillante, M.S., Zhang, Y.: Failure data analysis of a large-scale heterogeneous server environment. In: Proceedings of Dependable Systems and Networks (June 2004)
7. Tang, D., Iyer, R.K., Subramani, S.S.: Failure analysis and modelling of a VAX cluster system. In: Fault Tolerant Computing Systems (1990)
8. Xu, J., Kalbarczyk, Z., Iyer, R.K.: Networked Windows NT system field failure data analysis. In: Proc. of the Pacific Rim International Symposium on Dependable Computing (1999)
9. Schroeder, B., Gibson, G.: A large-scale study of failures in high-performance-computing systems. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN), Philadelphia, PA (June 2006)
10. Iyer, R.K., Rossetti, D.J., Hsueh, M.C.: Measurement and modeling of computer reliability as affected by system activity. ACM Transactions on Computing Systems 4(3) (1986)
11. Castillo, X., Siewiorek, D.: Workload, performance, and reliability of digital computing systems. In: 11th International Conference on Fault Tolerant Computing Systems (1981)
12. Oliner, A.J., Stearley, J.: What Supercomputers Say: A Study of Five System Logs. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN), Edinburgh, UK, June 2007, pp. 575–584 (2007)
13. Lan, Z., Li, Y., Gujrati, P., Zheng, Z., Thakur, R., White, J.: A Fault Diagnosis and Prognosis Service for TeraGrid Clusters. In: Proceedings of TeraGrid 2007 (2007)
14. Gujrati, P., Li, Y., Lan, Z., Thakur, R., White, J.: Exploring Meta-learning to Improve Failure Prediction in Supercomputing Clusters. In: Proceedings of International Conference on Parallel Processing, ICPP (2007)
15. Li, Y., Lan, Z.: Using Adaptive Fault Tolerance to Improve Application Robustness on the TeraGrid. In: Proceedings of TeraGrid 2007 (2007)

16. Lan, Z., Li, Y.: Adaptive Fault Management of Parallel Applications for High Performance Computing. *IEEE Transactions on Computers* 57(12), 1647–1660 (2008)
17. Oliner, A.J., Sahoo, R.K., Moreira, J.E., Gupta, M., Sivasubramaniam, A.: Fault-aware job scheduling for Bluegene/L systems. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium, IPDPS (2004)
18. Weaver, C., Austin, T.: A fault tolerant approach to microprocessor design. In: Proceedings of the International Conference on Dependable Systems and Networks, July 2001, pp. 411–420 (2001)
19. Austin, T.: DIVA: A reliable substrate for deep submicron microarchitecture design. In: Proceedings of the Thirty-Second International Symposium on Microarchitecture, November 1999, pp. 196–207 (1999)
20. Vijaykumar, T., Pomeranz, I., Cheng, K.: Transient fault recovery using simultaneous multithreading. In: Proceedings of the Twenty-Ninth Annual International Symposium on Computer Architecture, May 2002, pp. 87–98 (2002)
21. Oh, N., Shirvani, P.P., McCluskey, E.J.: Error detection by duplicated instructions in superscalar processors. *IEEE Transactions on Reliability* 51(1), 63–75 (2002)
22. Oh, N., Mitra, S., McCluskey, E.J.: ED4I: Error Detection by Diverse Data and Duplicated Instructions. *IEEE Transactions on Computers* 51(2), 180–199 (2002)
23. Slegel, T., et al.: IBM's S/390 G5 microprocessor design. *IEEE Micro* 19(2), 12–23 (1999)
24. Tullsen, D.M., Eggers, S.J., Levy, H.M.: Simultaneous multithreading: Maximizing on-chip performance. In: Proceedings of the Twenty-Second International Symposium on Computer Architecture, June 1995, pp. 392–403 (1995)
25. Rotenberg, E.: AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In: Proceedings of the Twenty-Ninth International Symposium on Fault-Tolerant Computing Systems, June 1999, pp. 84–91 (1999)
26. Sundaramoorthy, K., Purser, Z., Rotenberg, E.: Slipstream processors: Improving both performance and fault tolerance. In: Proceedings of the Thirty-Third International Symposium on Microarchitecture, December 2000, pp. 269–280 (2000)
27. Reinhardt, S.K., Mukherjee, S.S.: Transient fault detection via simultaneous multithreading. In: Proceedings of the Twenty-Seventh International Symposium on Computer Architecture, June 2000, pp. 25–36 (2000)
28. Qureshi, M.A., Mutlu, O., Patt, Y.N.: Microarchitecture-based introspection: A technique for transient-fault tolerance in microprocessors. In: Proceedings of International Conference on Dependable Systems and Networks, June 2005, pp. 434–443 (2005)
29. Parashar, A., Sivasubramaniam, A., Gurumurthi, S.: SlicK: slice-based locality exploitation for efficient redundant multithreading. In: Proceedings of the 12th Intl., conference on ASPLOS (2006)
30. Cooper, A.E., Chow, W.T.: Development of on-board space computer systems. *IBM Journal of Research and Development* 20(1), 5–19 (1976)
31. Jewett, D.: Integrity S2: A fault-tolerant Unix platform. In: Digest of Papers Fault-Tolerant Computing: The Twenty-First International Symposium, Montreal, Canada, June 25–27, pp. 512–519 (1991)
32. AT&T 5ESS™ from top to bottom,
<http://www.morehouse.org/hin/ess/ess05.htm>
33. AT&T Technical Staff. The 5ESS switching system. *The AT&T Technical Journal* 64(6), Part 2 (July-August 1985)
34. Avizienis, A.: Arithmetic error codes: Cost and effectiveness studies for Application in digital system design. *IEEE Transactions on Computers* 20(11), 1332–1331 (1971)

Out-of-Core Parallel Frontier Search with MapReduce

Alexander Reinefeld and Thorsten Schütt

Zuse Institute Berlin, Germany

Abstract. Applying the MapReduce programming paradigm to frontier search yields simple yet efficient parallel implementations of heuristic search algorithms. We present parallel implementations of Breadth-First Frontier Search (BFFS) and Breadth-First Iterative-Deepening A* (BF-IDA*). Both scale well on high-performance systems and clusters. Using the N -puzzle as an application domain, we found that the scalability of BFFS and BF-IDA* is limited only by the performance of the I/O system. We generated the complete search space of the 15-puzzle (≈ 10 trillion states) with BFFS on 128 processors in 66 hours. Our results do not only confirm that the longest solution requires 80 moves [10], but also show how the utility of the Manhattan Distance and Linear Conflicts heuristics deteriorates in hard problems. Single random instances of the 15-puzzle can be solved in just a few seconds with our parallel BF-IDA*. Using 128 processors, the hardest 15-puzzle problem took seven seconds to solve, while hard random instances of the 24-puzzle still take more than a day of computing time.

1 Introduction

MapReduce [2] is a parallel programming paradigm for processing large data sets that do not fit into main memory. The programmer specifies a *map* function that processes key/value pairs to generate a set of intermediate key/value pairs, and a *reduce* function that merges the key/value pairs with the same key. All work in *map* and *reduce* is done in parallel. The runtime system takes care of domain-independent tasks such as data partitioning, process scheduling, out-of-core data merging, process communication, synchronization, and automatic restart of crashed processes. We applied MapReduce to Breadth-First Frontier Search (BFFS) and Breadth-First Iterative-Deepening A* (BF-IDA*). Our algorithms run efficiently on massively parallel systems or clusters with distributed storage. By utilizing *all* parallel computing resources (processors, main memories and disks), they allow us to solve problems that were previously intractable. Using the N -puzzle as an initial application domain, we found that the scalability is only limited by the performance of the I/O system, that is, by the maximum number of open files and the file I/O throughput.

Fig. 1 illustrates frontier search with n parallel processes. Each process gets a bucket of nodes and applies the *map* function to generate successor nodes which are written into one of the n output buckets, depending on their signature. The MapReduce framework then merges all n^2 buckets into n buckets and the subsequent parallel *reduce* stage merges the operators of duplicate keys. The output is fed, as input, into the next *map* stage until a goal node is found or the solution space is exhausted.

The impetus to this work came from the observation that the *reduce* function in MapReduce implements a technique known as *delayed duplicate detection* [10][17] in

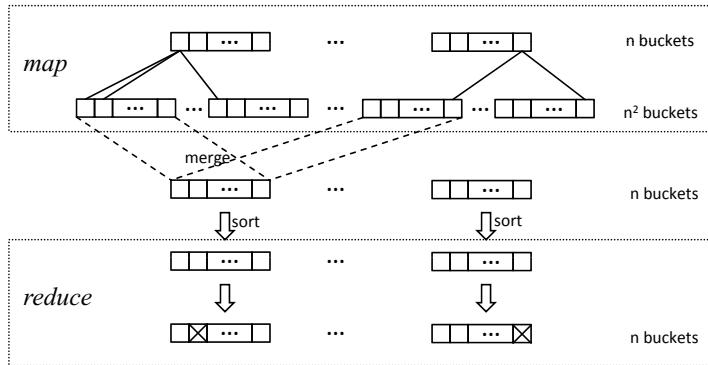


Fig. 1. Work flow of parallel frontier search with MapReduce on n processors

breadth-first or best-first searches. Hence, our algorithms can be regarded as parallel variants of Korf *et al.*'s Frontier Search [8][10][11], Zhou *et al.*'s Breadth-First Heuristic Search [16][15] and Edelkamp *et al.*'s External A* [4].

As a demonstrator application we chose the N -puzzle. Solving the hardest problem of the 15-puzzle took only seven seconds with BF-ID A^* on 128 processors, whereas the hardest random instances of the 24-puzzle still took more than a day of computing time.

In another experiment, we generated the complete search space ($16!/2 \approx 10^{13}$ states) of the 15-puzzle with BFFS and stored the resulting 80 TByte of position data on disk. This allowed us to analyze the utility of heuristic estimate functions in more detail than was previously possible¹. The following findings are discussed in Sec. 5:

- Breadth-first search eliminates 50% duplicates per search level (Fig. 4). This explains the relative poor efficiency of linear space searches like IDA*, which are not capable of detecting and removing duplicates.
- The *average* utility of the Manhattan heuristic improves linearly with the search depth, while it decreases for the best cases (optimal h) in depths ≥ 57 (Fig. 5a).
- The Linear Conflicts heuristic is useful in cases where Manhattan is weak, but it also deteriorates in depths ≥ 62 (Fig. 5b).
- The combined Manhattan and Linear Conflicts heuristic (Fig. 6) underestimates the true solution length on the average by 15 moves (Fig. 7).

2 Background

We consider search in implicit, undirected graphs with uniform edge cost. Implicit graphs are incrementally generated during the search process. They are described by

¹ Korf and Schultze [10] performed the first complete breadth-first search of the 15-puzzle, but they did not save the search graph. Their search took 28 days on two processors. Their implementation is more complex than ours (Fig. 2) and it is interesting to read how they had to cope with various hardware failures. This is done by the MapReduce framework in our case.

a start node, a set of operators that generate all successors of a given node, and a predicate that tests whether a node is a goal node. Several algorithms have been devised for finding optimal (shortest path) solutions.

Best-first searches store the visited nodes in two lists, an *Open* list of active nodes and a *Closed* list of nodes that have been expanded by generating all successors. In each step, an Open node with least cost is expanded, moved to the Closed list, and all successors that are not contained in Closed are inserted into the Open list. The A* best-first search algorithm [6] expands in each step an Open node n with the lowest cost $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the start node to the current node n and $h(n)$ is a cost estimate to the goal. When the current Open node is the goal node, A* terminates with a minimum cost solution, provided that the heuristic estimate h is *admissible*, that is, non-overestimating.

The Closed list serves two purposes: It is used to avoid duplicate-node expansions and to backtrack the solution path when a goal node is found. Because of its high storage requirements, A* is only applicable to small problem instances where the complete search graph fits into main memory. Without delayed duplicate detection (explained below), it is not feasible to store the Closed list on disk, because each newly generated node must be checked against the Closed list to avoid cycles or transpositions.

Iterative Deepening A (IDA*)* [7] is a space-efficient alternative to A*. It employs a series of depth-first searches with successively increased cost bounds. Like A*, IDA* finds optimal solutions. But in contrast to A*, it may re-examine a node several times because of transpositions, cycles, and the iterative re-expansion of shallower parts. In essence, iterative-deepening search trades space for time. IDA* is especially efficient in applications with cheap operator costs, such as the N -puzzle. Due to its depth-first approach, the algorithm is well suited for sequential execution, but it is difficult to parallelize [13]. Romein *et al.* [14] presented a table driven scheme which achieves linear scalability on a moderate number of processors by pushing work to processors.

Breadth-first search finds an optimal solution in graphs with unit edge cost by expanding a node in depth g only after all nodes of depth $g - 1$ have been expanded. It can be derived from A* by setting $h(n) = 0$. Compared to IDA*, which enumerates all shortest *paths*, breadth-first search enumerates all *nodes*, one depth after the other. Consequently, breadth-first search needs $O(w^d)$ storage space in trees of depth d and width w . For many practical applications, breadth-first search is too space-intensive, but its inherent data parallelism makes it attractive for parallel implementations.

3 Frontier Search

Frontier search [9][11] maintains an Open list of newly expanded nodes (the 'frontier'), but not a Closed list of previously visited nodes. To prevent the node expansion process from leaking back into the area of already visited nodes, enough nodes must be stored to form a boundary between the frontier and interior of the search graph. This can be done either by storing the previous frontier [16] or by memorizing for each node the operators that were used to reach it [9].

The first goal node found in the search process is optimal. Its solution path can, however, only be determined with a separate traceback function, because frontier search keeps only the frontier nodes but not their paths.

Frontier search is a general technique that can be applied to several search schemes, resulting in different algorithms:

1. *Breadth-first frontier search (BFFS)* [9] is the most simple form. It divides the search graph into layers, one for each depth, that are successively expanded. It can be used to explore complete search spaces, as shown in Sec. 4.
2. *Breadth-first heuristic search (BFHS)* [16] and *External A** [4] examine the nodes similarly to A* with the exception that the Open and Closed lists are organized in layers and previous layers are deleted to save memory. An upper bound on f^* is used to prune successors that cannot be on the optimal path. When a goal node is found, a divide-and-conquer traceback function determines the solution path through a previously stored intermediate node.
3. *Breadth-first iterative deepening A* (BF-IDA*)* [16] performs a breadth-first heuristic search with $h(n)$ of the start node n as an initial upper bound. If no solution is found, the bound is increased by the minimal cost of all unexpanded nodes, and the BFHS search is repeated. Note that, in contrast to BFHS, these upper bounds are optimal. Hence, BF-IDA* expands in the last iteration the same nodes as A*. We present a BF-IDA* implementation based on MapReduce in Sec. 6.

Both, BFFS and BFHS have been parallelized, albeit for a small number of processors only. Korf and Schultze [10] used six POSIX threads to hide the disk access latency of their BFFS. Their implementation took 28 days to generate the complete search space of the 15-puzzle. Zhang and Hansen [15] implemented BFHS for in-memory search on shared-memory processors. They solved random instances of the 15-puzzle on up to six CPUs. Compared to our approach, which runs on highly parallel systems, both implementations are inherently limited in their scalability.

3.1 Eliminating Duplicates

In many practical applications, the frontier list is too large to be kept in main memory. The newly generated successor nodes are then appended to a disk file. In a separate step, the duplicates in the file are merged with an external sort or hashing. This so-called *delayed duplicate detection* technique allowed Korf *et al.* [10][11] to explore the complete search space of the 15-puzzle for the first time [10].

However, it is not necessary to sort the whole frontier. The node space can be split into disjunct subsets, e.g., according to some characteristic like the position of the blank in the 15-puzzle. If there are enough splitting planes, the resulting sub-files are so small that they can be sorted in main memory. We use this *structured duplicate detection* [17] in our MapReduce implementation.

4 Breadth-First Frontier Search with MapReduce

Breadth-first frontier search (BFFS) is well suited for out-of-core execution, because there are no data dependencies within a frontier and nodes are not randomly accessed. To generate the complete search space, the following steps are performed²:

² Alternatively, to find a solution for a given position, we insert the position into the frontier list in step 0 and check against the goal state in step 3.

```

mapper(Position position, set usedMoves) {
    foreach((successor, move) ∈ successors(position))
        if (inverse(move) ∉ usedMoves)
            emit(successor, move);
}

reducer(Position position, set<set> usedMoves) {
    moves = ∅;
    foreach(move ∈ usedMoves)
        moves = moves ∪ move;
    emit(position, moves);
}

main() {
    front = {(start_position, ∅)};
    while (front ≠ ∅) {
        intermediate = map(front, mapper());
        front = reduce(intermediate, reducer());
    }
}

```

Fig. 2. Parallel breadth-first frontier search (BFFS) with MapReduce

- (0) Initialize the frontier list by inserting the start node.
- (1) For all nodes in the frontier list create the successors which do not undo operations and label them with the used operator.
- (2) Merge all duplicate nodes and label them with the union of the used operators.
- (3) Continue with step 2 unless the frontier is empty.

The first step iterates over the frontier nodes and creates successors. Because of delayed duplicate detection, the successor nodes can be simply appended to a new temporary list which becomes the frontier in the next iteration. In the second step, the successors are partially sorted to place duplicates adjacent to each other. Duplicates are then merged by linearly iterating over the nodes.

Fig. 2 shows a MapReduce implementation of BFFS for the N -puzzle. The code is simple, because the MapReduce framework organizes the parallel computation. The mapper gets a key/value pair consisting of a position (key) and a set of usedMoves (value) as input. It generates all successors and emits them together with the corresponding moves. For each unique position in the intermediate output, the reducer gets the position and all usedMoves, one for each duplicate. It merges the sets into one and emits the position with the combined usedMoves. The output of the reducer is fed, as input, into the next mapper until there are no more key/value pairs to process.

4.1 Implementation

For our first implementation, we used Hadoop³, an open-source implementation of MapReduce. Because of stability problems and too much overhead in storing the key/value pairs, we later settled on our own implementation using C++ and MPI. The

³ <http://hadoop.apache.org/>

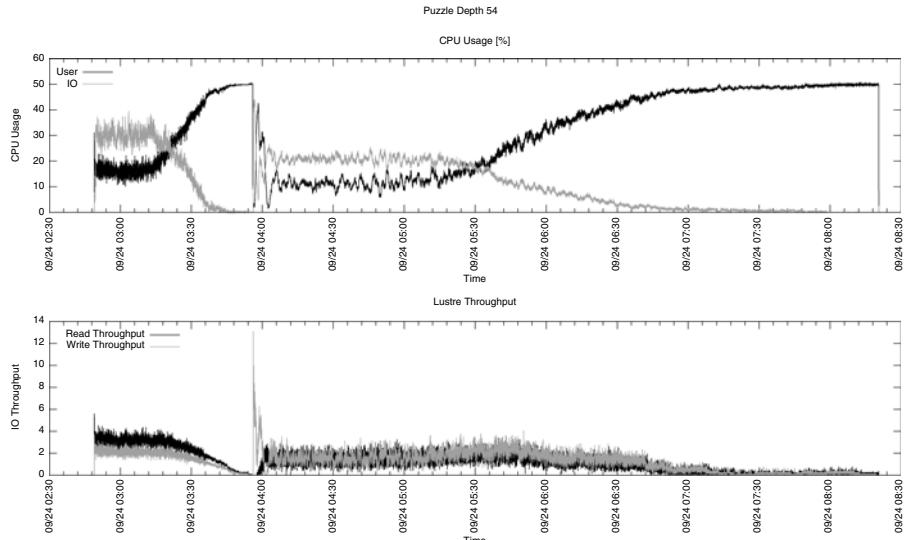


Fig. 3. CPU time (top) and I/O throughput (bottom) for map and reduce on the widest BFFS front

out-of-core data handling was implemented using STXXL [3], a C++ implementation of STL containers backed by disk storage.

We partitioned the search space into $n = 128$ buckets. Each key/value pair is stored in 64 bit. The key (position) contains the first 15 tiles using 4 bits each (60 bits); the 16th tile can be reconstructed from the available information. The value (usedMoves) is stored in the remaining 4 bits, where each bit represents one of the 4 possible moves up, down, left, or right.

5 Complete Solution of the 15-Puzzle with Parallel Breadth-First Frontier Search

Generating the complete solution space of the 15-puzzle took 66 hours on an SGI Altix XE250 with 32 nodes. Each node was equipped with two quadcore Intel Harpertown (3 GHz) and 64 GB main memory. The nodes were interconnected by a 4xDDR Infiniband network. We used only 128 of the available 256 cores because 128 cores saturated our I/O system and because our parallel Lustre file system was capable of handling only $128^2 = 16,384$ open files at a time.

Fig. 3 shows the CPU usage and I/O throughput of the map/reduce steps at the widest search front ($d = 54$). Most of the time was spent in the reduce function. The map and reduce phases can be easily distinguished in Fig. 3 as they are ended by MPI_Barrier which uses busy waiting to synchronize the processes. Note that we did not strive for maximum performance with our implementation. Rather, we directly translated the pseudocode shown in Fig. 2 into real code without any tuning.

Table 1. States per depth for the complete solution space of the 15-puzzle

d	map output	reduce output	d	map output	reduce output
0	0	1	41	105,232,854,740	83,099,401,368
1	2	2	42	148,081,981,988	115,516,106,664
2	4	4	43	203,644,257,664	156,935,291,234
3	10	10	44	273,637,877,374	208,207,973,510
4	24	24	45	358,812,754,412	269,527,755,972
5	54	54	46	458,895,663,682	340,163,141,928
6	108	107	47	571,810,278,100	418,170,132,006
7	214	212	48	693,606,328,846	500,252,508,256
8	454	446	49	818,215,994,990	581,813,416,256
9	966	946	50	937,634,709,510	657,076,739,307
10	2,012	1,948	51	1,042,659,315,592	719,872,287,190
11	4,060	3,938	52	1,123,608,988,412	763,865,196,269
12	8,122	7,808	53	1,171,775,867,988	784,195,801,886
13	16,156	15,544	54	1,180,827,850,290	777,302,007,562
14	32,338	30,821	55	1,171,775,867,988	742,946,121,222
15	63,794	60,842	56	1,073,945,899,830	683,025,093,505
16	125,460	119,000	57	965,058,046,960	603,043,436,904
17	244,358	231,844	58	831,765,594,568	509,897,148,964
18	475,468	447,342	59	684,759,208,366	412,039,723,036
19	914,426	859,744	60	538,359,358,490	317,373,604,363
20	1,755,070	1,637,383	61	401,797,523,128	232,306,415,924
21	3,327,004	3,098,270	62	285,131,794,258	161,303,043,901
22	6,278,248	5,802,411	63	190,679,390,020	105,730,020,222
23	11,702,798	10,783,780	64	120,772,653,396	65,450,375,310
24	21,684,174	19,826,318	65	71,454,350,152	37,942,606,582
25	39,674,570	36,142,146	66	39,922,745,752	20,696,691,144
26	72,079,114	65,135,623	67	20,599,350,216	10,460,286,822
27	129,231,886	116,238,056	68	10,019,242,068	4,961,671,731
28	229,705,420	204,900,019	69	4,426,822,772	2,144,789,574
29	402,601,352	357,071,928	70	1,840,952,666	868,923,831
30	698,293,484	613,926,161	71	677,567,466	311,901,840
31	1,193,224,016	1,042,022,040	72	234,051,602	104,859,366
32	2,014,400,784	1,742,855,397	73	68,291,438	29,592,634
33	3,346,429,688	2,873,077,198	74	18,421,746	7,766,947
34	5,482,511,216	4,660,800,459	75	3,758,728	1,508,596
35	8,827,837,774	7,439,530,828	76	683,042	272,198
36	13,992,014,012	11,668,443,776	77	70,550	26,638
37	21,766,319,560	17,976,412,262	78	8,168	3,406
38	33,266,838,564	27,171,347,953	79	180	70
39	49,831,763,900	40,271,406,380	80	34	17
40	73,199,529,058	58,469,060,820	Sum	15,716,295,466,894	10,461,394,944,000

We saved all positions with their used moves and distances to the goal state on disk. This occupies approx. 80 TBytes ($16!/2 * 8$ byte) of disk space. The data are split into 80 partitions, one for each depth. Each partition is again split into n buckets, each of them holding a fraction of the states and their corresponding used sets. With 128 buckets this results in a total of $80 \times 128 = 10,240$ files holding the complete 15-puzzle search space. The states are sorted within each file, which facilitates the retrieval.

Tab. 1 lists the number of generated nodes (map column) and explored nodes (reduce column) per depth. Summing all unique nodes gives $16!/2 = 10,461,394,944,000$ as it should be. The reduce column is the same as in [10], but additionally we list the total number of generated nodes in the map output. Comparing the two columns reveals an interesting result: one third of all generated nodes are duplicates! This explains (partly) the overhead of IDA*, which is capable neither of eliminating duplicates in the same depth nor of avoiding cycles in the graph. Note that the share of duplicates increases exponentially with the depth, as shown in Fig. 4. In the last search front ($d = 80$), only 17 of the 34 generated nodes are unique.

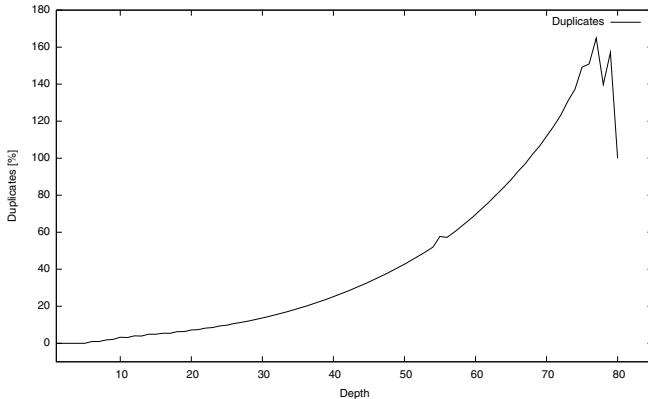


Fig. 4. Duplicates per depth (*map/reduce – 1*)

5.1 Retrieving Optimal Solutions

Given a random position, we can not only tell how many steps are needed to solve the position (as in [10]), but also determine the solution path. This is done iteratively, starting at the deepest level f^* : For each node along the path, we compute the position modulo the total number of buckets (128) to find the bucket number b of the given position. Then we retrieve from the b -th bucket in the given depth the configuration and its `usedMoves`-list, which tells us which move(s) must be undone to get one step closer to the initial position. By repeatedly undoing a move and checking the next lower depth for the resulting position, we traverse the graph backwards until we reach the initial position. The resulting path (resp. graph) is optimal.

Note that there may be more than one `usedMoves` stored in the used set. They can be used to generate a backward graph containing all shortest paths of move transpositions. The backward graph can be generated in parallel using MapReduce—analogously to the described forward search.

5.2 How Good are Manhattan Distance and Linear Conflicts Heuristics?

We used the data of the complete search space of the 15-puzzle to analyze the quality of the Manhattan Distance (MD) and Linear Conflicts (LC) [5] heuristics. Fig. 5a shows the quality of MD for all positions sorted by depth. The vertical bars show the best (top) and the worst (bottom) estimate. The line in the middle gives the average quality over all estimates of that depth. As can be seen, the MD quality increases linearly with the depth, but the discrepancy to the true value (diagonal) gets more pronounced in deeper trees. This indicates a decreasing utility in positions with long solution paths. Note that for all depths > 56 there is not even a single case where MD gives the correct answer.

Fig. 5b shows the same plot for the LC heuristic [5]. As can be seen, LC is of little use in positions with very short and very long solution paths: In the short paths, most

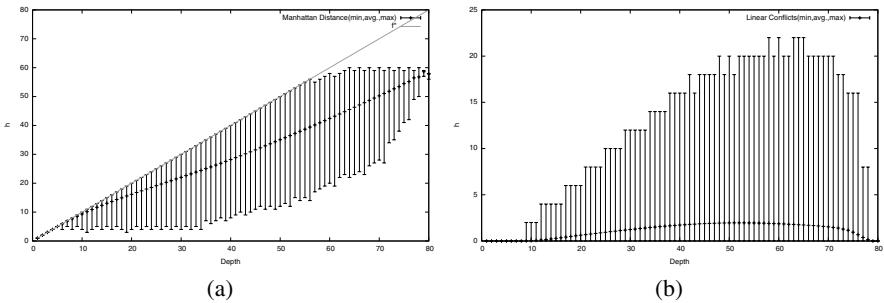


Fig. 5. Quality of (a) Manhattan Distance and (b) Linear Conflicts per depth

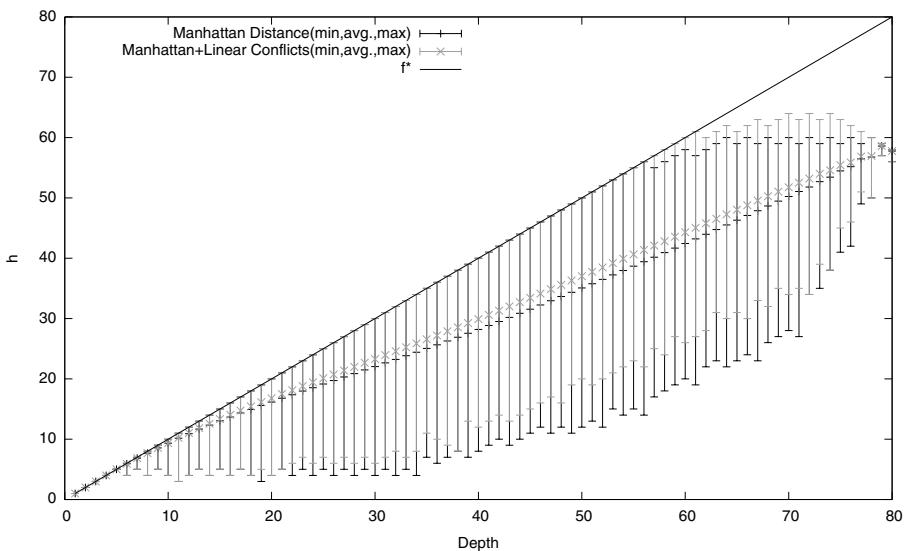


Fig. 6. Quality of the combination of Manhattan Distance + Linear Conflicts

tiles are already in their goal position, while in the long paths, they are not in the same row and/or column, and hence, no linear conflicts occur.

The combination of both heuristics, shown in Fig. 6 gives a clear advantage over using a single one. In all depths up to 61 there exist cases where the combination returns the optimal value $h = f^*$ on the actual solution length (see top of the grey bars meeting the diagonal). Even in the worse estimates (bottom of the grey bars) the combination is much better than using a single heuristic (black bars).

Fig. 7 shows a histogram of the quality of the heuristics for all $16!/2$ positions. The left and the middle curves show the quality of MD and MD+LC, respectively, and the right curve shows the distribution of the true distance f^* . The given data suggests an average heuristic error of 15, that is, the combined MD+LC heuristic underestimates the true solution length f^* by 15 moves.

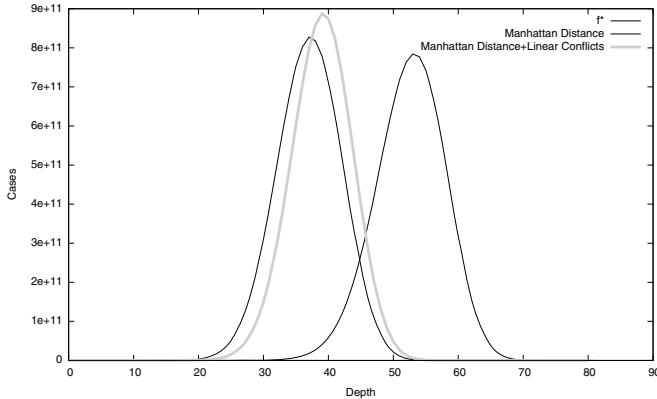


Fig. 7. Histogram of the solution lengths predicted by MD (left), MD+LC (middle), and true solution length f^* (right) for all $16!/2$ configurations

6 Breadth-First Heuristic Frontier Search with MapReduce

MapReduce can be only be applied not only to breadth-first frontier search as presented in Sec. 4, but also to search algorithms that use a heuristic estimate to quickly guide the search process into the most ‘promising’ parts of the solution space. Heuristic search algorithms are designed to find an optimal solution to a given position rather than to solve a complete search space. Typical examples are A* and IDA*. Both can be modified to expand the nodes in layers with the parallel MapReduce framework.

We present two variants: One that uses iterative-deepening with MapReduce (breadth-first iterative-deepening A* (BF-IDA*)) and one that is based on A* (breadth-first heuristic frontier search).

6.1 Breadth-First Iterative-Deepening A*

BF-IDA* has the same iterative-deepening search pattern as IDA*, but it employs a breadth-first node expansion in each iteration. Consequently, it never examines a node twice within an iteration.

The program code of BF-IDA* in Fig. 8 can be derived from BFFS (Fig. 2) by adding a single program line to the mapper function and by calling the mapper and reducer in a loop. With this added program line, the mapper only emits successors n with $f(n) \leq \text{thresh}$. In an outer loop, the search depth thresh is iteratively incremented from one iteration to the next.

With its breadth-first node expansion, BF-IDA* never generates a node twice in an iteration. In the last iteration it generates exactly the same nodes as A* does. All previous iterations are needed to determine the optimal upper bound $\text{thresh} = f^*$. Fig. 9 shows the node expansions in one of the seventeen hardest 15-puzzle configurations with $f^* = 80$. The widest search front contains four billion nodes (approx. 33 GB).

```

mapper(Position position, set usedMoves) {
    foreach((successor, move) ∈ successors(position))
        if(inverse(move) ∉ usedMoves)
            if(g + 1 + h(successor) ≤ thresh) // added line
                emit(successor, move);
}

reducer(Position position, set<set> usedMoves) {
    moves = ∅;
    foreach(move ∈ usedMoves)
        moves = moves ∪ move;
    if(position == goal-position)
        solved = true;
    emit(position, moves);
}

main() {
    thresh = h(n);
    solved = false;
    while(!solved) {
        front = {(n, ∅)};
        g = 0;
        while (!solved && front ≠ ∅) {
            intermediate = map(front, mapper, thresh, g);
            front = reduce(intermediate, reducer());
            g++;
        }
        thresh++;
    }
}

```

Fig. 8. Parallel Breadth-First IDA* (BF-IDA*) with MapReduce

Note that the search effort is dominated by the node expansions in the last iteration which must also be examined by A* (however, A* cannot be used for this instance because of the storage and maintenance overhead of its Closed list).

Fig. 10 compares the MapReduce BF-IDA* to IDA* with the MD and MD+LC heuristics. For each search depth (except $d = 80$), we picked 128 random positions and solved them with both methods and both heuristics. As can be seen, BF-IDA* with MD+LC searches up to 32 times fewer nodes than IDA* with MD. Interestingly, the combined MD+LC heuristic has a greater effect on the overall performance than the elimination of duplicates in the iterations by BF-IDA*.

Comparing IDA* and BF-IDA* with the same heuristic, BF-IDA* most often outperforms IDA* because it never re-examines a node in the same iteration. However, anomalies may occur when IDA* finds the solution 'by chance' in the leftmost part of the search space. It then immediately stops the node expansion process and returns the optimal solution. This explains the zigzag appearance of the right part of the graphs, which is attributed to the increasing impact of node ordering in the last IDA* iteration.

6.2 Breadth-First Heuristic Frontier Search

BFHS [16] and External A* [4] are heuristic search schemes similar to A*. Their layered Closed and Open lists can also be processed efficiently with the described

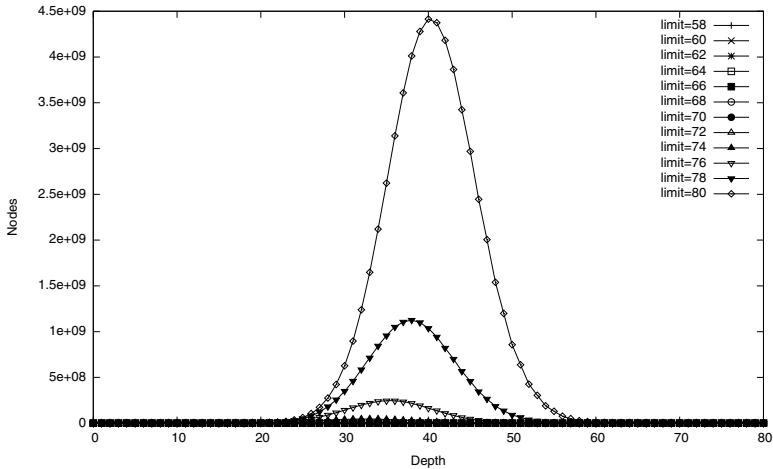


Fig. 9. BF-IDA* node expansions per iteration in one hard instance ($d = 80$) with MD+LC

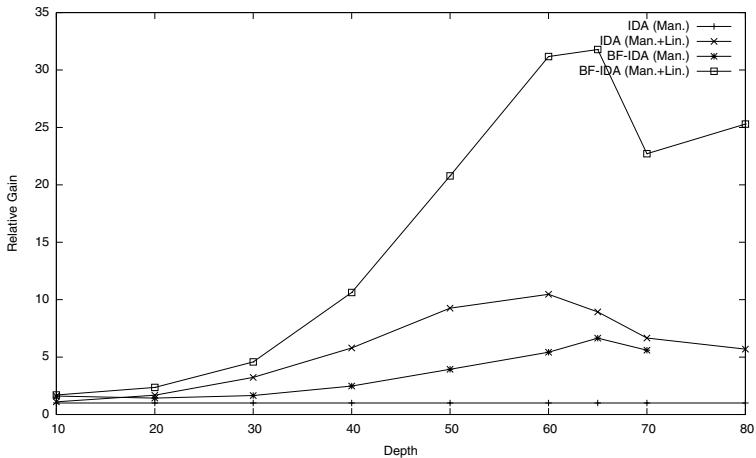


Fig. 10. BF-IDA* node expansions relative to sequential IDA*

MapReduce approach. However, in contrast to BF-IDA*, the layers cannot be generated and processed in a single map/reduce step, because a newly generated successor could have the same f value as its predecessor and, hence, must be further expanded. This happens when, in one move, an increase of g is offset by a decrease of h .

Confirming earlier results of Zhou et al. [16], our BFHS implementation with MapReduce yielded also a 30% reduction of the node expansions as compared to BF-IDA*. This can be seen in Fig. 9 where BFHS only expands the nodes of the last iteration (limit = 80), but not the previous ones.

7 Conclusions

We applied the MapReduce programming model to breadth-first frontier search (BFFS), breadth-first iterative-deepening A* (BF-IDA*), and breadth-first heuristic frontier search (BFHS). The code is efficient, fault-tolerant and easy to implement because the MapReduce framework takes care of all domain-independent tasks. With its efficient utilization of distributed resources (CPU, main memory, and disks), MapReduce solves single problem instances or complete search spaces that would otherwise be intractable.

Our implementations were built on previously published algorithms for frontier search [8][10][11], breadth-first heuristic search [16][15], and external A* [4]. But they are conceptually simpler (Figs. 2 and 8) and they run on highly parallel systems with distributed storage. Closest to our work is probably *Transposition Driven Scheduling* [14] which integrates work scheduling with the use of memory tables. But again, we believe that our MapReduce approach is easier to implement, because of its generic approach that hides the complexity of the parallel execution environment. We conjecture therefore, that the MapReduce programming paradigm will encourage a revival of various breadth-first searches and perhaps even of the search algorithms used in adversary games.

With the complete search space of the 15-puzzle stored on disk, our future work will focus on the detailed evaluation of heuristic estimate functions [5] and the systematic computation of memory tables [1][14] for deriving improved heuristics.

Acknowledgements. Aside from the scientific goals, the algorithms served as a throughput benchmark for our Lustre file system. We thank the North-German Supercomputing Alliance (HLRN) for providing computer time and disk storage capacity.

References

1. Culberson, J., Schaeffer, J.: Searching with pattern databases. In: McCalla, G.I. (ed.) Canadian AI 1996. LNCS, vol. 1081, pp. 402–416. Springer, Heidelberg (1996)
2. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: OSDI (2004)
3. Dementiev, R., Kettner, L., Sanders, P.: STXXL: Standard template library for XXL data sets. In: 13th Annual European Symp. on Algorithms, pp. 640–651 (2005)
4. Edelkamp, S., Jabbar, S., Schrödl, S.: External A*. In: Biundo, S., Frühwirth, T., Palm, G. (eds.) KI 2004. LNCS (LNAI), vol. 3238, pp. 226–240. Springer, Heidelberg (2004)
5. Hansson, O., Mayer, A., Yung, M.: Criticising solutions to relaxed models yields powerful admissible heuristics. Information Sciences 63, 207–227 (1992)
6. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Sys. Sci. Cyber. 4(2) (1968)
7. Korf, R.E.: Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence, 97–109 (1985)
8. Korf, R.E.: Divide-and-conquer bidirectional search: First results. In: IJCAI 1999, pp. 1184–1189 (1999)
9. Korf, R.E., Zhang, W.: Divide-and-conquer frontier search applied to optimal sequence alignment. In: AAAI 2000, pp. 910–916 (2000)

10. Korf, R.E., Schultze, P.: Large-scale parallel breadth-first search. In: AAAI 2005, pp. 1380–1385 (2005)
11. Korf, R.E., Zhang, W., Thayer, I., Hohwald, H.: Frontier search. J. ACM 52(5) (2005)
12. Korf, R.E.: Linear-time disk-based implicit graph search. J. ACM 55(6) (2009)
13. Reinefeld, A., Schnecke, V.: AIDA* – Asynchronous Parallel IDA*. In: Canadian Conf. Artificial Intelligence, pp. 295–302 (1994)
14. Romein, J.W., Bal, H.E., Schaeffer, J., Plaat, A.: A Performance Analysis of Transposition-Table-Driven Work Scheduling in Distributed Search. IEEE Trans. Parallel and Distributed Systems 13(5) (2002)
15. Zhang, Y., Hansen, E.A.: Parallel breadth-first heuristic search on a shared-memory architecture. In: Workshop on heuristic search, memory-based heuristics and their appl. (2006)
16. Zhou, R., Hansen, E.A.: Breadth-first heuristic search. In: 14th Intl. Conf. on Automated Planning and Scheduling, ICAPS 2004 (2004)
17. Zhou, R., Hansen, E.A.: Structured duplicate detection in external memory graph search. In: AAAI 2004, pp. 683–688 (2004)

Hybrid MPI-Cell Parallelism for Hyperbolic PDE Simulation on a Cell Processor Cluster

Scott Rostrup and Hans De Sterck

Department of Applied Mathematics, University of Waterloo, Ontario,
N2L 3G1, Canada

Abstract. We show how a numerical simulation method for nonlinear hyperbolic partial differential equation (PDE) systems on structured grids with explicit timestepping can be implemented efficiently for the Cell processor and for clusters of Cell processors. We describe memory layout, communication patterns and optimization steps that are performed to exploit the parallel architecture of the Cell processor. A second layer of Message Passing Interface (MPI) parallelism is added to obtain a hybrid parallel code that can be executed efficiently on Cell clusters. Performance tests are conducted on a Cell cluster, and the Cell performance is compared with x86 performance (Xeon). Compared with single-core Xeon performance, the Cell processor obtains significant speed-ups of 60x for single precision calculations, and 20x for double precision. In a chip-to-chip comparison, the Cell code is 14x faster than a 4-core Xeon (using pthreads) in single precision, and 5x faster in double precision. Parallel cluster scaling results were hampered by a relatively slow interconnect on our test system, but overall our study shows how Cell clusters can be used efficiently for simulating nonlinear hyperbolic PDE systems.

1 Introduction

The Cell Broadband Engine Architecture (CBEA) developed jointly by IBM, Sony, and Toshiba is a microprocessor design focussed on maximizing computational throughput and memory bandwidth while minimizing power consumption. The first implementation of the CBEA is the Cell processor, which has already been used successfully in several large scale scientific clusters [12], notably Los Alamos National Laboratory's Petaflop scale system Roadrunner [4]. Encouraged by these successes we explore the Cell processor for finite-volume simulation of nonlinear hyperbolic partial differential equation (PDE) systems. In particular, we describe a Cell simulation code that solves the Shallow Water Equation nonlinear PDE system numerically on structured Cartesian grids in two dimensions (2D) using explicit time integration.

The strengths and challenges of the Cell architecture derive from its unconventional heterogeneous multi-core architecture. We first describe an optimized code that exploits the parallelism present within a single Cell processor. We then add an additional top layer of parallelism that utilizes the Message Passing Interface (MPI) library. This leads to a high-performance hybrid code with

two levels of parallelism that can be used efficiently on a Cell processor cluster. The performance of the optimized Cell code is compared to x86 performance (Xeon), and parallel scaling tests are performed on a Cell cluster provided by SHARCNET (Shared Hierarchical Academic Research Computing Network: www.sharcnet.ca).

This paper is structured as follows. We first describe the hardware characteristics of the Cell processor and the cluster we use for our tests in Section 2. This is followed by a brief description of the Shallow Water PDE system and the numerical solution technique in Section 3. Section 4 describes memory layout, communication patterns and optimization steps for a single Cell processor, and Section 5 presents parallel scaling results for a single blade with two Cell processors, compared with a dual-processor x86 Xeon blade server. Section 6 describes parallel scaling results on a Cell cluster composed of multiple Cell blades, and Section 7 discusses conclusions and future work.

2 Hardware Description

The Cell processor ([76]) achieves a high floating point throughput to power ratio by using a heterogeneous multi-core architecture. The Cell design may be thought of as a network on a chip with different cores specialized for different computational tasks (Figure 1). Since the Cell is designed for high computational throughput applications, eight of its nine processor cores are vector processors, called Synergistic Processing Elements (SPE). The other core is a more conventional (and relatively slow) Power 5 Central Processing Unit (CPU), called the Power Processing Element (PPE). It runs the operating system and is suitable for general purpose computing. However, in practice its main task is to coordinate the activities of the SPEs.

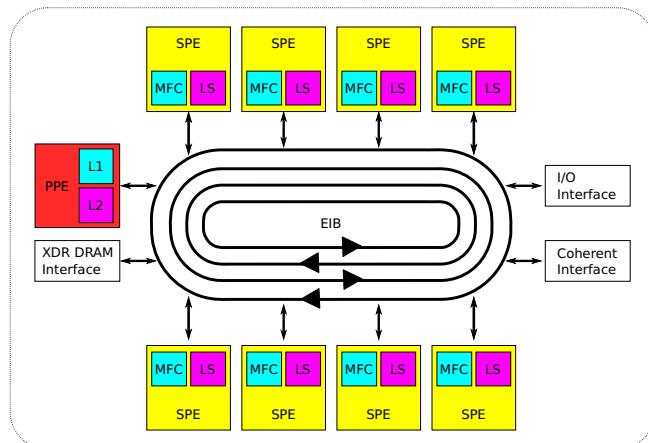


Fig. 1. Hardware diagram of the Cell processor

Communication on the chip is carried out through the Element Interconnect Bus (EIB). It has a high bandwidth (204.8 GB/s) and connects the PPE, SPEs, and main memory access through a four channel ring topology, with two channels going in each direction. For main memory the Cell uses Rambus XDR DRAM memory which delivers 25.6 GB/s maximum bandwidth on two 32 bit channels of 12.8 GB/s each.

2.1 Power Processing Element (PPE)

The PPE is a 64 bit processor with a 512KB L2 cache that uses IBM's POWER PC (PPC) instruction set. It may run any typical computer application using the PPC instruction set and operate as a general purpose CPU. However this is not a particularly effective use of the Cell's computational resources and in most Cell-specific applications the role of the PPE is limited to the coordination of the SPEs.

2.2 Synergistic Processing Element (SPE)

The Synergistic Processing Element (SPE) has a 3.2GHz Single Instruction Multiple Data (SIMD) processor (called Synergistic Processing Unit (SPU)) that operates on 128-bit wide vectors which it stores in its 128 128-bit registers. The SPU is a dual issuing processor, and, provided that the two instructions satisfy certain constraints, it can issue two instructions per cycle. For instance, it can issue a load operation at the same time as a multiplication instruction.

Each SPE has 256KB of on-chip memory called the Local Store (LS). The SPU draws on the LS for both its instructions and data: if data is not in the LS it has no automatic mechanism to look for it in main memory. All data transfers between the LS and main memory are controlled via software-controlled direct memory access (DMA) commands. Each SPE has a memory flow controller that takes care of DMAs and operates independently of the SPU. DMAs may also transfer data directly between the local stores of different SPEs.

The SPU has only static branch predicting capabilities and has no other registers besides the 128-bit registers. It supports both single and double precision floating point instructions. However, hardware support for transcendental functions is only available for single precision sqrt and division. Double precision transcendental functions are evaluated in software.

2.3 Programming the Cell Processor

Programming complexity is one of the challenges to using the Cell Processor, as its design sacrifices programming ease for increased computational throughput. For in-depth information on programming the Cell, the interested reader is referred to Scarpino's excellent book [2] or IBM's programming guide [3].

The Cell processor may be programmed in either C/C++ or fortran, using a common library-provided API. A program consists of one PPE executable and

potentially many SPE executables. The SPE executables are included into the PPE executable as libraries at compile time and may then be executed on an SPE using pthreads and library function calls.

Data is moved between main memory and the LS on each SPE by using library-provided memory get and put functions. Each takes an address in main memory, an address in the LS, and a size in bytes. A get sends data to the LS, while a put sends data to main memory. Both of these functions may only be called from and SPU executable, have a maximum transfer size of 16KB, and the addresses and sizes must be aligned to 16 byte boundaries.

2.4 SHARCNET Cell Cluster

The SHARCNET Cell Cluster system ‘prickly’ that we used for code development and testing has the following specifications:

- eight QS22 blade servers
- each QS22 blade has two PowerXCell 8i processors @ 3.2GHz in a dual-socket configuration that gives access to 16GB of shared memory
- the eight QS22 blades have a GigE Interconnect

In addition, the system has four more traditional dual-Xeon server blades. Note that this setup is markedly different from Roadrunner nodes, since a relatively slow GigE link interconnects the Cell and Xeon blades in prickly, while Roadrunner has PCIe x8 links between QS22 and Opteron blades and Infiniband between nodes. Due to the relatively slow interconnect in prickly, we do not utilize the Xeon blades for MPI message passing, but use the Cell PPEs for this purpose (see below).

2.5 Comparison with x86 Xeon Processors

Throughout this paper, we compare the Cell code performance with performance of an equivalent code on quad-core Xeon processors (2.66GHz).

3 Hyperbolic PDE Simulation Problem

Our code computes numerical solutions of the Shallow Water equations, which are given by

$$\frac{\partial}{\partial t} \begin{bmatrix} h \\ hu \\ hv \end{bmatrix} + \frac{\partial}{\partial x} \begin{bmatrix} hu \\ hu^2 + \frac{gh^2}{2} \\ huv \end{bmatrix} + \frac{\partial}{\partial y} \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{gh^2}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad (1)$$

where h is the height of the water, g is gravity, and u and v represent the fluid velocities. The gravitational constant g is taken to be one in the test simulations reported in this paper. The Shallow Water system is a nonlinear system of hyperbolic conservation laws [5], and given an initial condition, a two-dimensional

(2D) domain and appropriate boundary conditions, it describes the evolution in time of unknown quantities $h(x, y, t)$, $u(x, y, t)$ and $v(x, y, t)$. We discretize the equations on a rectangular domain with a structured Cartesian grid, and evolve the solution numerically in time using a finite-volume numerical method with explicit time integration [5]. In what follows we write $U = [h \ hu \ hv]^T$, and we update the solution in each grid cell (i, j) using an explicit difference method that depends only on U at the previous time step n according to the update formula

$$U_{i,j}^{n+1} = U_{i,j}^n - \frac{\Delta t}{\Delta x} (F_{i+\frac{1}{2},j}^n - F_{i-\frac{1}{2},j}^n) - \frac{\Delta t}{\Delta y} (G_{i,j+\frac{1}{2}}^n - G_{i,j-\frac{1}{2}}^n). \quad (2)$$

Here, F and G stand for numerical approximations to the fluxes of Eq. (1) in the x and y directions, respectively. Rather than (2), we use a dimensional splitting approach

$$\begin{aligned} Q_{ij}^* &= Q_{ij}^n - \frac{k}{\Delta x} \left(F_{i+\frac{1}{2},j}^n - F_{i-\frac{1}{2},j}^n \right), \\ Q_{ij}^{n+1} &= Q_{ij}^* - \frac{k}{\Delta y} \left(G_{i,j+\frac{1}{2}}^* - G_{i,j-\frac{1}{2}}^* \right), \end{aligned} \quad (3)$$

which allows us to treat the x and y updates in two separate sweeps (see [5], p. 444). We use a limited second-order accurate expression for the numerical fluxes F and G ([5], p. 121, Eqs. (6.59)-(6.60)), and utilize a Roe Riemann solver ([5], p. 481). The resulting difference method is highly accurate, and the arithmetic density per grid point is high, which makes this algorithm a good candidate for acceleration on the Cell processor, along with the structured nature of the grid data.

Note that we have chosen a relatively simple set of hyperbolic equations for this code description and performance study paper. However, more complicated hyperbolic systems, including the compressible Euler and Magnetohydrodynamics equations which are widely used for fluid and plasma simulations, can be approximated numerically by the same methods, and extension of our approach from 2D to 3D is also not difficult. The approach and results of our paper thus carry over directly to a wide class of important fluid and plasma simulation problems.

3.1 Test Problem

The test problem used for the simulations in this paper has initial condition

$$\begin{aligned} h(x, y, 0) &= \frac{1}{2} (x + y) + 1, \\ u(x, y, 0) &= v(x, y, 0) = 0, \end{aligned}$$

on a unit square domain with lower left corner situated at $(0, 0)$. Boundary conditions are perfect walls [5].

4 Optimized Code for Single Cell Processor

The parallelism provided by the eight SPEs on the Cell processor is exploited by dividing the computational task of evolving each grid cell forwards in time into data-parallel sub-tasks. This is done by dividing the computational grid into blocks of grid cells whose size is determined by the size of the LS on the SPEs. Workloads are assigned to each SPE by giving each one a collection of grid blocks to be updated. In order to efficiently send blocks into and out of the local stores a distributed memory layout is adopted in the main memory of the Cell blades. In addition, SIMD vector parallelism is used to accelerate the individual cell update computational kernel using SPU vector intrinsics. In addition to SIMD parallelism several other optimizations are performed on the compute kernel.

4.1 Memory Layout

Two optimizations are done to minimize the effect of memory latency on performance. The first optimization is motivated by the fact that few large DMA transfers may be executed more efficiently than many small DMA transfers. Thus, when transferring a subblock out of a larger array it is inefficient to start a new DMA transfer for every row of the array. A better approach is to store the full array as a sequence of blocks, exactly as in an MPI distributed memory implementation (Figure 2). By doing this each block may be transferred into and out of a LS in far fewer DMA operations. The cost of doing this is that updated grid cell values must be communicated between blocks through the use of a halo of ghost cells surrounding each block. These halo communications may be efficiently implemented through the use of exchange buffers (see below). The net result of the distributed layout is far fewer DMA transfers than in a single array layout.

The second optimization exploits the ability of the SPU to operate independently of its memory flow controller. This means that the SPU's calculations

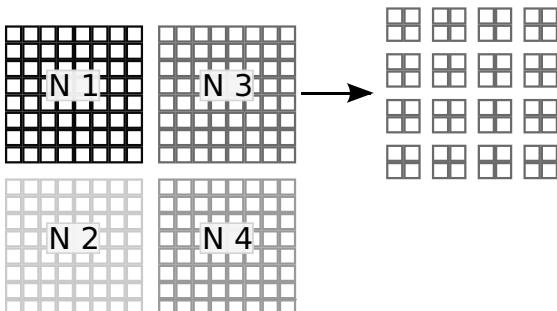


Fig. 2. The blocks on the left represent an MPI distributed memory layout, the blocks on the right are the internal representation of Node 3's grid in local store sized blocks

may be overlapped with memory transactions through the use of two buffers to hold data. Given two buffers, A and B, in the LS, while calculations are being performed on buffer A, data is being transferred into and out of buffer B. Once both computation and data transfer are complete, their roles reverse and buffer B is used for calculation and buffer A for data transfer. This double-buffering combined with the previous optimization virtually eliminates memory latency delay from the overall computation time.

4.2 Neighbour Communication

The halo ghost cell communications introduced by the distributed memory layout are implemented in two separate ways and their performance is compared. The first is a naive implementation that utilizes the PPE to transfer all of the halo data in the blade's main memory, and the second uses the SPEs and additional exchange buffers.

PPE Neighbour Exchange. In this implementation, at each time step the PPE waits until all SPEs have completed updating all of their grid blocks. The PPE then moves updated data into the ghost cells of each grid block in the main blade memory while the SPEs stall. This implementation is inefficient for a number of reasons:

1. It fails to use the thread-level parallelism by not using the SPEs.
2. The PPE has a lower bandwidth to main memory than an individual SPE.
3. It causes the SPEs to stall unnecessarily.

SPE Neighbour Exchange. The solution to these problems is to use the SPEs to directly send updated data into exchange buffers stored with the neighbouring block in the main blade memory. When the block is next loaded into a local store, the exchange buffer is brought in as well. The SPE then shuffles the buffer values into the ghost cells and computation proceeds as normal. In this way the time taken to do neighbour communications is hidden behind calculations and no longer affects performance (Table I).

Table 1. Performance of the PPE and SPE neighbour exchange schemes for a typical simulation. Execution times in seconds to do neighbour exchange and computation, respectively. Notice that in the SPE scheme compute time increases slightly, but the exchange time is hidden almost entirely behind calculations.

		PPE Exchange	SPE Exchange
Single Precision	Exchange	19.3	0.0
	Compute	5.88	6.02
Double Precision	Exchange	51.55	0.0
	Compute	22.53	22.82

4.3 SPE Kernel Optimizations

Timing results for three different SPE kernel implementations are presented in Table 2. Each implementation uses the data layout mechanism described above, including the SPE exchange. The first kernel is simply a naive porting of the x86 CPU grid cell update kernel code to the Cell architecture. The two other kernels represent two different ways of dealing with unaligned data in the SPE LS, and both are optimized for the Cell hardware.

Naive. In the naive kernel, the grid cell update compute kernel is directly copied from the x86 CPU version with minimal alterations. This does not give very good performance as it ignores the SIMD nature of the SPEs and significant amounts of branching are present inside of nested loops (the SPE has very poor performance on branch prediction).

SIMD, Shuffle. The changes made in this version involve making use of the SPU vector intrinsics to SIMDize all floating point operations. Two difficulties arise while doing this. The first is that when updating in the x direction, neighbouring cell data is not aligned to 16 byte boundaries. (All data in the SPE local store is accessed in 16-byte blocks that are located at 16-byte boundaries.) In this implementation, data in the SPE local store is shuffled into the correct alignment directly before it is needed for calculation, using an efficient SPU intrinsic shuffling instruction. The second difficulty is that the branching statements used in the flux limiter calculations must be eliminated by using vector comparisons and selections (they work on four floating point numbers for single precision, and two for double precision). The code then calculates all branches for all entries and selects the correct branch at the end, which is much faster than the SPU's native static branch prediction. SPU intrinsics used for vector comparisons and selections include `spu_cmplt` and `spu_sel`.

SIMD, Transpose. This implementation is similar to the previous one except that, instead of shuffling data in the interior loops, the entire grid block is transposed prior to calculation (using shuffling).

Table 2 shows timing results for the various kernel optimizations, compared to the serial code running on one Xeon core. All versions are compiled with high optimization settings: the x86 version is compiled with -O3 with the Intel icc compiler, and the Cell versions are compiled using IBM's ppuxlc and spuxlc, with optimization settings -O3 or -O5, whichever gives better performance. Note that the icc compiler optimizes for x86 Streaming SIMD Extensions (SSE) instructions. The naive Cell implementation is already faster than the serial x86 CPU implementation, and excellent speedups of approximately 60x and 20x are obtained for the optimized Cell code, for single and double precision, respectively. For the Cell, single precision is at least twice as fast as double precision, because the SIMD vectors contain four single precision numbers or two double

Table 2. The runtimes (seconds) of the various kernel optimizations are compared against each other and a reference serial x86 CPU implementation running on one Xeon core. The grid and domain variables for the test case used are given in Section 3. Grid size is $1,000 \times 1,000$. The naive implementation is already faster than the serial x86 CPU implementation. The fastest cell kernel gives close to a 60x speed-up in single precision and a 20x speed-up in double precision. All Cell runs use all eight SPEs.

Kernel	Single Precision		Double Precision	
	time (s)	speed-up (x)	time (s)	speed-up (x)
one Xeon core	353.93	1.00	425.26	1.00
Naive Cell	115.82	3.06	150.01	2.83
SIMD Cell, Shuffle	5.95	59.48	22.81	19.17
SIMD Cell, Transpose	6.59	53.71	25.28	16.82

precision numbers. In addition, double precision transcendental functions, including square roots and divisions, are implemented in software, which increases the number of floating point operations executed. It is therefore no surprise that the timing results in Table 2 show that our double precision Cell kernel is about four times slower than the single precision kernel. Note that the penalty for using double precision is much smaller on the Xeon.

5 Parallel Scaling on a Single QS22 Blade

Here we briefly discuss the parallel scaling of the optimized Cell code (with Shuffle optimization) on a single QS22 blade. On a QS22 blade, up to all 16 SPEs of the two Cell processors can be steered by one of the two PPEs. (The two Cell processors have access to the shared blade memory via a dual-socket configuration.) This allows the user to utilize the full processing power of the two Cell processors on the QS22 blade just by using the single-Cell code with a larger number of SPEs, and without the need for an additional layer of threading or MPI implementation. In Table 3 we show how excellent nearly linear scaling is obtained for a speed-up test on a single QS22 blade (total problem size is kept constant throughout the test).

Table 4 shows a comparison of runtime on one QS22 blade with two Cell processors, and on one server with two quad-core Xeon processors that have access to shared memory via a dual-socket configuration. The Xeon code uses pthreads, so no MPI is required. This comparison is interesting because it shows how a Xeon blade with all cores in use compares to a fully utilized Cell blade. It also allows to compare the Xeon and Cell processors on a chip-by-chip basis. Again, we conclude that a Cell blade is significantly faster than a Xeon blade for single precision (about 14x), and still about 5x faster for double precision.

6 Parallel Scaling on a Cell Cluster with MPI

In this section we describe parallelization of the optimized Cell code (with Shuffle optimization) via MPI, and give scaling results on the prickly Cell system. An

Table 3. Parallel scaling (constant total problem size) of optimized Cell code (with Shuffle optimization) on a single QS22 blade (runtime in seconds) for the test problem of Section 3. The scaling is almost perfectly linear. Grid size is $1,000 \times 1,000$.

SPEs	Single Precision		Double Precision	
	time (s)	speed-up (x)	time (s)	speed-up (x)
1	47.62	1.00	181.24	1.00
2	23.85	2.00	91.23	1.99
4	11.96	3.98	45.63	3.97
8	6.02	7.91	22.82	7.94
16	3.00	15.87	11.41	15.88

Table 4. Runtime comparison on a QS22 blade with two Cell processors, and a dual-socket server with two quad-core Xeon processors. Grid size is $1,000 \times 1,000$.

	Single Precision		Double Precision	
	time (s)	speed-up (x)	time (s)	speed-up (x)
Kernel				
2 quad-core Xeons	43.50	1.00	58.13	1.00
2 Cell processors	3.09	14.07	11.39	5.10

additional upper level of parallelism is added to the Cell code, in which the PPE units send MPI messages to logically neighbouring nodes. Data are assigned to the parallel nodes in a grid layout, with each MPI node getting a rectangular subdomain of data. This gives each node (up to) four neighbouring nodes with whom it must communicate. Communications are done after each time step, sending two layers of updated ghost cell data to the neighbouring nodes. Since the data on the nodes are stored in a distributed fashion (the subdomain of each node is subdivided in grid blocks that are treated by the SPEs), the two rows or columns of grid cell data must be gathered from the grid blocks and collected into a buffer in order to be sent to the neighbouring node. Once neighbours have exchanged data, the ghost cell data must be distributed back into the grid blocks.

In Table 5 we show scaling results for the MPI code on the prickly cluster, which has eight QS22 blades with two Cell processors each, and a GigE interconnect. The computation time (comp) scales well, but the inter-node communication time (comm) does not scale well, and since it makes up a significant part of the total time, the total scaling is also sub-optimal. Scaling for double precision calculations is somewhat better since communication takes up relatively less time there. The limitation of having a slow GigE interconnect becomes apparent here. Due to the slow interconnect, we chose to let the PPEs handle the MPI traffic, rather than letting faster processors on separate blades perform the MPI communication (like on Roadrunner nodes). The PPEs are slow, and the network is slow as well. Both these factors contribute to the large communication time, and the sub-optimal scaling results that follow from it. A faster interconnect (like Infiniband) would alleviate this and would allow to use faster processors on

Table 5. Scaling results for the MPI Cell code on the prickly cluster. All times are in seconds. Each of the two PPEs on a parallel node steer eight SPEs. Grid size is $10,000 \times 10,000$.

nodes	Single Precision				Double precision			
	comm	comp	total	speed-up	comm	comp	total	speed-up
1	21.45	299.76	321.21	1.00	25.18	1132.91	1158.09	1.00
2	22.63	150.36	172.99	1.73	29.24	566.56	595.80	1.90
4	36.57	75.48	112.05	2.94	43.29	285.46	328.75	3.45
8	30.40	42.68	73.08	4.01	36.01	163.36	199.37	5.68

separate blades to perform the MPI communication. It has been demonstrated successfully on Roadrunner, and similar systems, that good scalability can be obtained in this way, but unfortunately the hardware that we have currently available does not allow us to explore these improvements yet.

7 Conclusions and Future Work

We have shown how a numerical simulation method for nonlinear hyperbolic PDE systems can be implemented efficiently for the Cell processor and for clusters of Cell processors. We have described memory layout, communication patterns and optimization steps that were performed to exploit the parallel architecture of the Cell processor. A second layer of MPI parallelism was added to obtain a hybrid parallel code that can be executed efficiently on Cell clusters composed of multiple QS22 blades. Performance tests were conducted on SHARCNET's Cell cluster system prickly and the Cell performance was compared with Xeon performance.

Compared to single-core Xeon performance, significant speed-ups were obtained of 60x for single precision calculations, and 20x for double precision. In a chip-to-chip comparison, the Cell is 14x faster than a 4-core Xeon (with pthreads) in single precision, and 5x faster in double precision. While our parallel cluster scaling results were hampered by a relatively slow interconnect on our test system, the overall results of our study show that Cell clusters can be used for efficiently simulating nonlinear hyperbolic PDE systems on structured grids with explicit timestepping.

We have ongoing work on hyperbolic PDE simulation codes for clusters with GPU accelerators, and plan to report on this in the near future. Future work will also include extension of our approach to bodyfitted adaptive multi-block codes in 3D, which allow for simulation of real 3D problems with more complex geometry. Extending our work to unstructured grids or implicit time integration would be more challenging, since those typically require unstructured data and/or nonlocal connectivity. Mixed-precision calculations are also an interesting option, due to the excellent performance Cell processors feature for single precision calculations.

Our results support indications that clusters with heterogeneous multi-core architectures may become increasingly important for scientific computing applications in the near future, especially also if one considers their typically low power requirements [4] and the fact that heterogeneous multi-core architectures may scale more easily to large numbers of on-chip cores than homogeneous chips with full x86 cores.

Acknowledgements

This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca). We thank Hugh Merz and other members of the SHARCNET technical staff for their expert advice and technical help. Additionally we want to thank Markus Stuermer from the University of Erlangen-Nuernberg, for providing answers to many of our questions about the Cell processor.

References

1. <http://www.green500.org>
2. <http://www.top500.org>
3. Arevalo, A., Matinata, R.M., Pandian (Raj), M., Peri, E., Ruby, K., Thomas, F., Almond, C.: Programming the Cell Broadband Engine Architecture: Examples and Best Practices. IBM (2008)
4. Barker, K.J., Davis, K., Hoisie, A., Kerbyson, D.J., Lang, M., Pakin, S., Sancho, J.C.: Entering the petaflop era: The architecture and performance of roadrunner. In: SC 2008 (2008)
5. LeVeque, R.J.: Finite volume methods for hyperbolic problems. Cambridge University Press, Cambridge (2002)
6. Pham, D.C., Aipperspach, T., Boerstler, D., Bolliger, M., Chaudhry, R., Cox, D., Harvey, P., Harvey, P.M., Hofstee, H.P., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Masubuchi, Y., Pham, M., Pille, J., Poslusny, S., Riley, M., Stasiak, D.L., Suzuoki, M., Takahashi, O., Warnock, J., Weitzel, S., Wendel, D., Yazawa, K.: Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. IEEE Journal of Solid-State Circuits 41(1), 179–196 (2006)
7. Scarpino, M.: Programming the Cell Processor: For Games, Graphics, and Computation. Prentice Hall PTR, Upper Saddle River (2008)

BW-DCache: An Inexpensive, Effective and Reliable Cache Solution in a SAN File System

Chengxiang Si^{1,2}, Xiaoxuan Meng^{1,2}, and Lu Xu¹

¹ Institute of Computing Technology, Chinese Academy of Sciences,
100190 Beijing, China

² Graduate School of the Chinese Academy of Sciences
100049 Beijing, China
chengxiangsi@gmail.com

Abstract. In order to enhance scalability of Blue Whale Cluster File System (BWFS), this paper analyses the bottleneck of scalability of BWFS in non-linear editing application, and presents a solution that clients employ local disk storage as cache on clients. We called *BW-DCache* (*BWFS Cache* based Disk medium). Moreover, we carefully choose the client cache strategy and replacement algorithm according to data access pattern. By this, clients can get as much data from local cache as possible, avoid random accesses to storage servers and efficiently reduce the load of storage servers. Therefore, the scalability of the whole system is significantly enhanced. The experiment result shows that our solution can enhance the scalability of the Blue Whale System by 80% at most. Moreover, we adopt some optimizations to improve system performance further.

Keywords: Cache, BWFS, Scalability.

1 Introduction

With the development of cluster and networking technologies, network storage system has been recently resorted to solve the I/O performance bottleneck problem inherent in cluster systems. As a key component of network storage system, the cluster file system has evident advantages over traditional file system, such as performance, capacity, data-sharing and scalability, etc. But in practice, the bandwidth storage devices provide doesn't linearly grow with the increasing number of clients which leads to bad QoS and restricts the scalability of cluster file systems. Therefore, how to fully utilize the bandwidth of networked storage device to provide better QoS and scalability is the key problem needed to be solved.

Blue Whale Cluster File System [1] (BWFS) is based on SAN architecture and adopts Out-Of-Band transfer mode which separates metadata from data path. It provides shared service at file level to nodes in cluster and now has been widely applied to some domains, such as data analysis, image processing, non-linear editing environment, etc. The system consists of application server cluster, metadata server cluster, and storage server cluster and administrator server. Fig.1 shows the architecture of

BWFS. Metadata servers are responsible for metadata management, storage servers provide storage service and administrator server takes charge of monitoring system status and management jobs.

In order to enhance QoS and scalability of BWFS, the paper analyses the bottleneck of scalability of BWFS in non-linear editing application, and presents a solution that clients employ local disk storage as cache on clients. We called *BW-DCache* (*BWFS Cache* based Disk medium). Moreover, we carefully choose client cache strategy and replacement algorithm according to data access pattern. By this, clients can get as much data from local cache as possible, avoid random accesses to storage servers and efficiently reduce the workload of storage servers. Therefore, the scalability of the whole system is significantly enhanced.

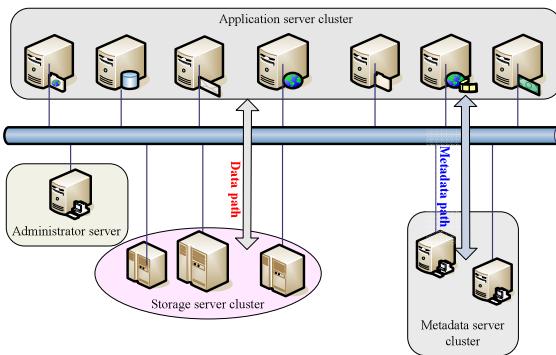


Fig. 1. The architecture of BWFS. The system consists of application server cluster, metadata server cluster, and storage server cluster and administrator server.

The paper is organized as follows. The next section analyses the data access pattern of non-linear editing application. Section 3 presents how to choose client cache strategy and cache replacement algorithm. Section 4 presents the system implementation. Section 5 shows system optimization. We present our experimental methodology in section 6 and conclusion in section 7.

2 Data Access Pattern of Non-linear Application

Non-linear editing is a kind of application which mainly process digital video, image, cartons by computer technology to finish film-making.

In this environment, application nodes consist of editing nodes, producing nodes, auditing nodes. In the working process, some editing nodes are responsible for processing material files, one producing node for making program files and one or two auditing nodes for verifying program files. The number of editing nodes supported by the system indicates the scalability of the system; therefore, improving the number of client nodes (in the following sections, clients represent “editing nodes”) which the system efficiently supports at the most is the key to enhance the system scalability.

2.1 Data Access Pattern of Editing Nodes

We get file-level trace of editing nodes in a real non-linear editing environment of some TV station. The trace, which is filtered through client page cache, represents IO load sent to storage servers. Its trait is shown in Table 1. It includes 3982299 operations and has a range of 271 minutes. It can represent the data access trait of editing nodes.

Table 1. Trait of the trace

Length	Footprint	I/Os	Read	write
271min	4.4GB	3982299	99.76%	0.24%

An editing node has two storage access levels, including client page cache and storage server. If a request misses in page cache, it must access storage server. By analyzing the trace, we get data access pattern of editing nodes.

1. For reading from local page cache, the amount of sequential read ranges from 5M to 8M and the size of a request is about 512k.
2. For reading from storage server, the amount of sequential read ranges from 3M to 10M and the size of a request is about 64k. Moreover, there are many file property queries operations between two consecutive reads.
3. Write operations are very few. The amount of sequential write ranges from 1M to 2M and the size of a request is from several bytes to hundreds of bytes.
4. The percent of write data amount is 0.24% of total data access amount.
5. 66% of read data from storage server is repeated.

According to the analysis above, if we add extra cache level (relative to client page cache) in editing nodes, the access times to storage server will decrease and the load of server will also fall down. Therefore, the number of clients which the server supports will increase and the scalability of the system is enhanced. What's more, it is worth notice that from clients' view, the extra cache level is second-level cache relative to client page cache; from servers' view, it behaves as client cache.

3 Choice of Cache Strategy and Replacement Algorithm

Cache is one of the main methods to improve the performance of networked storage system. There are two different kinds of mediums as cache. One is volatile, such as memory; the other is non-volatile, such as disk, NVRAM.

3.1 Cache Strategy

According to the analysis above, we have known that write operations are very few and write data amount is 0.24% of total data access amount. Moreover, write operation may introduce data consistency problem. As a result, the implementation cost of write buffer may be more than benefit which brings. So we only implement read data buffer at client and adopt “write through” to write operations.

Meanwhile, 66% of read data from storage server is repeated. In the trace, the size of this part is about 3.01G, so it is usually larger than memory size of a client. Moreover, the non-linear editing environment in which we collect trace employs the format of DV whose requirement for bandwidth is 25Mb/s. With the development of HDTV which requires 100Mb/s, the size of data set that is accessed repeatedly is 4 times larger than that of DV. So usually the memory size of a client is very insufficient.

In order to finish editing work better, the response time must be no greater than 10ms. The response time of modern disk meets the demand of editing nodes whose maximum response time is limited to 10ms. Furthermore, clients employing local disk partition as cache not only meet the demand of cache capacity, but also don't increase much more cost. At present, the price of disks is only about \$0.3/GB compared to \$11/GB of memory. In addition, disks are persistent so that there is no data loss, when power is suddenly off. According to the analysis above, we employ disk as cache medium and only implement read data buffer.

3.2 Cache Replacement Algorithm

In order to choose proper replacement algorithm, we first analyze the data access locality of clients. According to the locality property, we choose proper algorithm for our system. We mainly analyze temporal locality and access frequency to get access locality property of clients.

3.2.1 Temporal Locality

We borrow the concept in [12] to analyze temporal locality. A *reference sequence* is a numbered sequence of temporally ordered accesses to a cache. The *temporal distance* is the number of distinct accesses between two accesses to the same block in the reference sequence. For example, in the reference sequence ABCDBAX, the temporal distance from A1 to A2 is 5 and the temporal distance from B1 to B2 is 3. Therefore, the larger the temporal distance is, the weaker the temporal locality is. A temporal distance histogram shows the number of correlated accesses (accesses to the same block) for various temporal distances.

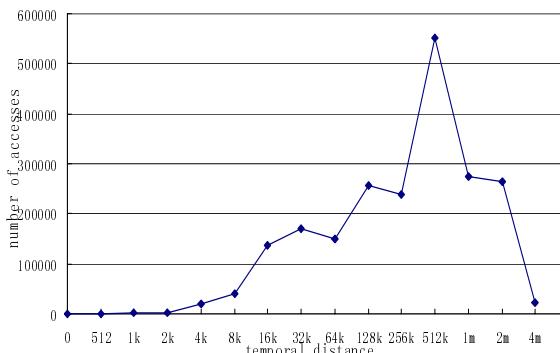


Fig. 2. Temporal distance distribution of the trace. The x-axis represents temporal distance, and the y-axis represents the number of correlated accesses.

Fig. 2 shows that 99% of the correlated accesses have a temporal distance of 4k or greater. If the size of a data block is 4k, the temporal distance of 99% of the correlated accesses is 16M or larger, exhibiting weaker temporal locality. It is clear that, if the size of cache is smaller than 16M, the LRU replacement algorithm tends to perform poorly, because most blocks do not stay in the cache long enough for subsequent correlated accesses.

A good replacement algorithm for caches should retain blocks that reside in the “hill” portion of the histogram for a longer period of time. In this paper, “time” means logical time, measured by the number of references. For example, initially, time is 0; after accesses *ABC*, time is 3. We call the beginning and end of this “hill” region minimal distance (or *minDist*) and maximal distance (or *maxDist*), respectively. A good replacement algorithm should keep most blocks in this region for at least *minDist* time.

3.2.2 Access Frequency

In order to understand the frequency distributions of the trace, we examined the relationship between access distribution and block distribution for different frequencies.

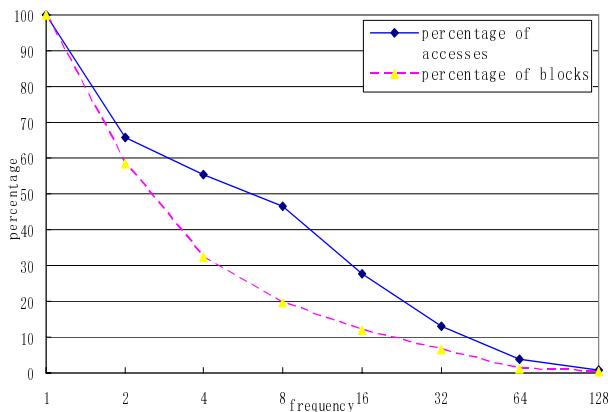


Fig. 3. Access frequency distribution of the trace

Fig. 3 shows, for a given frequency f , the percentage of total number of blocks accessed at least f times. It also shows the percentage of total accesses to those types of blocks. Notice that the number of blocks accessed at least i times includes blocks accessed at least j times ($j > i$). This explains why all the curves always decrease gradually. The access percentage curves decrease similarly for the same reason.

Fig. 3 shows that the access percentage curves decrease more slowly than the block percentage curves, indicating that a large percentage of accesses are to a small percentage of blocks.

Specially, 58% of the blocks have been accessed at least twice and contribute to 66% of the accesses. However, only 32% of the blocks have been accessed at least four times, but contribute to 56% of the accesses. Therefore, there is a need to differentiate blocks accessed at least four times from those accessed only two or three times

to make the former stay in cache long enough before they are evicted. In other words, the replacement algorithm should give different priorities for blocks with different frequencies.

According to the analytical result above, we summarize the access pattern with the following two properties: 1) large reuse distance of correlated accesses; 2) unevenly distributed access frequency among blocks.

Property one shows weak locality, so LRU would tend to behave poorly. According to property two, there is a need to differentiate blocks accessed at least four times from those accessed only two or three times to make the former stay in cache long enough before they are evicted.

3.2.3 Comparison of Replacement Algorithm

According to the analytical result above, a good replacement algorithm appropriate for the application should have the following three properties.

1. Minimal lifetime: the blocks that are frequently accessed should stay in buffer cache for at least $minDist$ time.
2. Frequency-based priority: blocks should be prioritized based on their access frequencies; consequently, the blocks are replaced according to their priorities.
3. Temporal frequency: the blocks were accessed frequently in the past, but have not been accessed for a relatively long time should be replaced.

We choose replacement algorithms which have these three properties, including MQ[12],2Q[13] and ARC[14]. Meanwhile, in order to compare, we also introduce LRU. We develop a simulator which implements these four replacement algorithms and replay the trace. According to the result of the simulation, we get their hit ratio and choose the optimal algorithm from them.

In the simulator, MQ adopts optimal parameters given by [12]. In the reference, there are eight LRU queues with different access frequencies and the history buffer is set to be four times of the cache size and the lifetime parameter is set to maximal temporal distance. 2Q employs optimal parameters suggested by [13] in which Kin is set to 30% of the cache size and $Kout$ is set to half of the cache size. ARC is adaptive so that there is no need to adjust parameters. The test result is shown in Table 2.

Although 2Q, MQ and ARC all have time complexity of $O(1)$, ARC has the smallest constant. Moreover, ARC only maintain the metadata for 2 times of cache size, however, MQ need the metadata for 5 times of cache size. The space overhead of ARC is the smallest. What's more, ARC is adaptive and can self-tune with variety of

Table 2. Hit ratio of different cache size

Cache Size(MB)	ARC(%)	LRU(%)	MQ(%)	2Q(%)
128	12.35	4.30	10.23	9.87
256	17.28	7.22	16.12	15.72
512	24.44	10.09	20.17	21.38
1024	41.35	28.38	38.53	37.12
2048	54.60	47.15	53.91	52.75
4096	65.25	64.17	65.10	65.12

temporal locality and frequency of data access. 2Q and MQ must adjust optimal parameters according to different applications. Fig.4 shows that LRU perform worst due to the weak locality of non-linear editing application. 2Q and MQ have similar hit ratio and ARC perform best. According to the analysis above, we employ ARC as our replacement algorithm of clients.

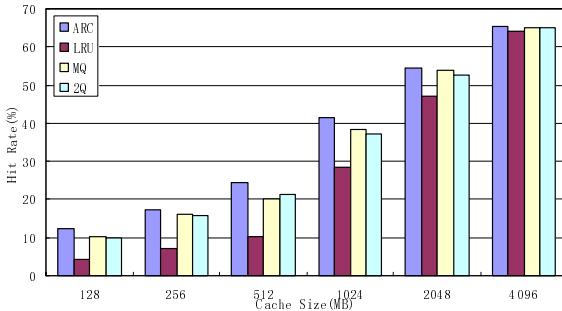


Fig. 4. Hit ratio of different algorithms with different cache sizes

4 Implementation of *BW-DCache*

4.1 Design Strategies of *BW-DCache*

According to the access pattern of the application and the result of the simulation above, we make several design strategies as follows:

1. We implement *BW-DCache* at file system level. By this way, we can make flexible cache strategies to files with special types; furthermore, it is beneficial to maintain file data consistency.
2. Keep data persistent. The non-volatile trait of disk maintains data persistent easily. When the client machine restarts, it can get data from local *BW-DCache* by re-building manageable information; consequently, there is no need to read repeated data again from the storage server.
3. The system adopts ARC algorithm.
4. The system implements read data buffer at client and adopts “write through” to write operations.
5. The buffer granularity is data block. According to the data access pattern, the size of data set accessed frequently is 3.01G and the number of the related files is 16.8% of total number. If the granularity is an entire file, it not only wastes cache space, but also increases the access overhead of the storage server.
6. The size of a buffer block is set to 64K. Due to the performance gap between random and sequential operations of disk medium, we try to make logical consecutive blocks in physical consecutive positions to improve access performance. If the size of a buffer block is too small, the amount of metadata will be large. Moreover, current replacement algorithms designed for memory care nothing about positions of

buffer blocks, which will increase data discontinuity on disk. On the contrary, if the size of a buffer block is too large, the accuracy of algorithms will be dropped, which will lead to the algorithm distortion. Among IO requests that clients submit, the proportion of 4KB, 16KB, 32KB, and 64KB is 3.3%, 1.5%, 1.4% and 77% respectively. Therefore, according to the analysis above, we select 64k as the size of a buffer block.

4.2 System Design of *BW-DCache*

We develop the system in windows platform which consists of algorithm management module, memory buffer module, data flush module, disk cache module.

The function of algorithm management module is to implement and manage ARC algorithm. Memory buffer module is responsible for management of buffer data in memory and mapping information. The function of data flush module is to decide when to flush data and what need to be flushed. Disk cache module is responsible for managing data buffer; moreover, the metadata resides in the fixed region of cache disk so that the client can recover quickly after restart. The index table is used to locate cache block, which employs linked hash table. Fig. 6 shows the structure of it, and “BitMap” indicates whether the data is valid or not.

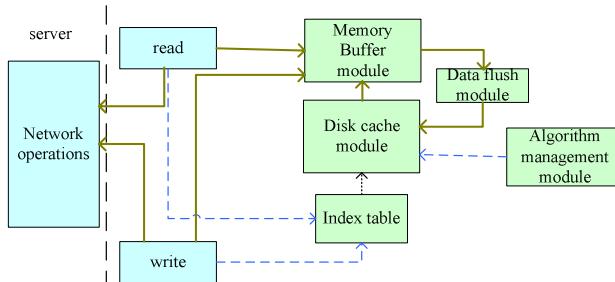


Fig. 5. The architecture of *BW-DCache*

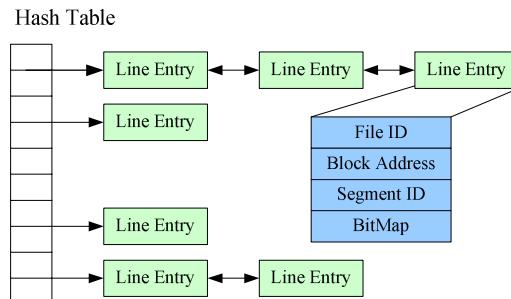


Fig. 6. The structure of the index table

4.3 Data Consistent

Because the BWFS protocol is stateless which is similar to NFS[5], the file system supports weak semantic consistency. We adopt two strategies----updating in chance and updating of time-out mechanisms. Updating in chance means that when fetching the mapping information between file blocks and physical blocks from the metadata server, the client actively checks “change time” of the request file. If the time changes, the relevant mapping information in cache is invalidated. If not, the time stamp is changed only. Time-out means that the client actively checks whether the mapping information is valid every fixed intervals. By combining the active updating with the passive one, to some extent, semantic consistency of file system is guaranteed.

4.4 Read and Write

In this section, we will illustrate read process and write process of BW-DCache briefly.

4.4.1 Read

1. Query the index table to check if the request blocks hit in BW-DCache.
2. If hit, read data from cache according to the index.
3. In 2nd step, if there are still some blocks missing in one request, a new request will be made and then is forward to the storage server. Goto 5th step.
4. If miss, the request is redirected to the storage server.
5. If there are free blocks in cache, the new data blocks are copied into cache. Or, discard the new blocks.

4.4.2 Write

1. Query the index table to check if the request blocks hit in BW-DCache.
2. If hit, write data to cache according to the index.
3. Forward the request to the storage server

5 System Optimization

5.1 Replacement Algorithm Optimization

As analysis in section 4.1, we set 64k as the size of a buffer block, which is larger than the minimum size (4KB) of IO request. In this case, the algorithm may appear to distort. Because the operation unit of the algorithm is a buffer block, if the size of IO request is smaller than 64K, this will lead to discrimination between history access records and current access information. For example, a client sends two requests continuously and the size of every request is 4KB. Moreover, two requests are both in the same disk buffer block and don't overlap. As a result, two requests should miss. However, due to the request size smaller than 64K, from the algorithm's view, two requests both access the same disk buffer block. Therefore, the former request behaves to miss, but the latter will hit. According to the algorithm, the buffer block will

be moved to high frequency list. But, in fact, the buffer block was accessed only once, and should keep in low frequency list. For the worst scenario, if requests are completely sequential, the current algorithm will make all buffer blocks in high frequency list. As a result, the scan-resistant property of ARC will lose.

Because the minimum size of a request and a buffer block in the replacement algorithm is fixed, we will modify the replacement algorithm to avoid the distortion. According to analysis above, the reason for the algorithm distortion is to indiscriminately deal with IO requests with different sizes. For this reason, we should modify the algorithm, when an IO request size is smaller than a buffer block size.

The details of modification as follows:

1. The request block in $T1$

a) If the size of a request is smaller than the size of a buffer block (64K) and the request data intersects with the according buffer block, the buffer block will be moved to MRU position of $T1$.

b) If the size of a request is smaller than the size of a buffer block (64K) and the request data doesn't intersect with the according buffer block, the buffer block keeps in place.

2. The request block in $T2$

a) If the size of a request is smaller than the size of a buffer block (64K) and the request data intersects with the according buffer block, the buffer block keeps in place.

b) If the size of a request is smaller than the size of a buffer block (64K) and the request data doesn't intersect with the according buffer block, the buffer block will be moved to MRU position of $T1$.

3. The request block in $B1$

a) If the size of a request is smaller than the size of a buffer block (64K) and the request data intersects with the according buffer block, the buffer block will be moved to MRU position of $T1$ and the value of P keeps unchanged.

b) If the size of a request is smaller than the size of a buffer block (64K) and the request data doesn't intersect with the according buffer block, the buffer block keeps in place and the value of P keeps unchanged.

4. The request block in $B2$

a) If the size of a request is smaller than the size of a buffer block (64K) and the request data intersects with the according buffer block, the buffer block will be moved to MRU position of $T1$ and the value of P keeps unchanged.

b) If the size of a request is smaller than the size of a buffer block (64K) and the request data doesn't intersect with the according buffer block, the buffer block keeps in place and the value of P keeps unchanged.

When the size of an IO request is smaller than that of a buffer block, the buffer block with low frequency may be moved to high frequency list. The rationale behind the modification is that employing more rigorous conditions confines a buffer block entering into high frequency list from low frequency list to prevent cache pollution.

Table 3 depicts the result of replacement algorithm optimization. Compared with the non-optimization version, the algorithm after optimization can improve hit ratio effectively.

Table 3. Comparison of hit ratio of non-optimization and optimization of ARC

Cache size	512MB	1024MB	2048MB	4096MB
Non-optimization	25.7%	43.2%	54.6%	65.6%
optimization	32.4%	49.4%	59.3%	66.4%

5.2 Cache Management Strategy Optimization

Compared with memory, the response time of disk medium is larger. Our system employs traditional buffer management strategy. When cache misses, system tries to buffer data. As a result, when the load of storage servers is very heavy, our system behave vey well. However, when the storage server is not relatively busy, the response time of read from the storage server may be smaller than that of read from the local BW-DCache. Consequently, the system performs unobtrusively. We think this is because the performance discrepancy between disk and network is not large enough.

We modify our cache management strategy. In particular, clients record the response time of reading from cache and storage servers respectively. According to response time, system decides if the request will read from cache, when it hits. If the response time of reading from cache is larger than reading from storage servers, the ensuing requests will be forwarded to storage servers directly. By this way, system can adapt the load fluctuation of storage servers effectively to achieve better performance.

6 Test and Evaluation

We employ read response time to evaluate our system, when multiple clients concurrently access to the storage server. For the fixed number of clients, if the response time with BW-DCache is smaller than that without BW-DCache, or, for the same response time, if the number with BW-DCache which the server supports is larger than that without BW-DCache, we conclude that our solution enhances the scalability of the storage server.

6.1 Test Methodology

We respectively replay trace at clients with BW-DCache and those without BW-DCache. Meanwhile, we record the average read data time. When trace replaying is finished, we compute the average read response time of clients. When clients concurrently access the same file in the storage server, server cache may include data of the file. In order to eliminate the affect of storage server cache, we simulate the same environment on the server for every client separately. Therefore, when replaying the trace, every client can access its own files only. Experiment setting is described in Table 4.

6.2 Performance Test

Table 4. Experimental setup

	AS(client)	SN(storage server)	MS(metadata a server)
CPU	Intel® Xeon(TM) 2.80G	Intel® Xeon(TM) 3.20G	Celeron ® 3.33G
memory	DDR 1GB	DDR 512M	DDR 512M
network	Gigabits	Gigabits	Gigabits
os	Windows Xp SP2	Linux -2.6.17	Linux-2.6.17
storage	IDE 80G	SATA 200G*2	

6.2.1 Concurrent Access Performance

In this test, we compare concurrent access performance of BW-DCache with that of no BW-DCache. The size of cache is set to 4.4G equal to the size of data set that a client accesses.

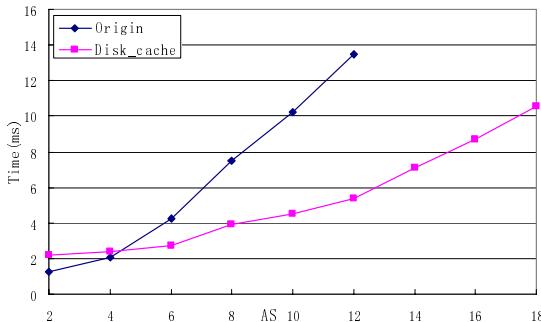
Maximal read response time is limited to less than or equal to 10ms, or the editors will feel uncomfortable. In order to achieve the best effect, we test multiple clients until read response time reaches 10ms. Due to limit of our experimental conditions, we only test the cases with 14 client nodes at the most. We adopt the least square method to fit the case with 16 nodes and 18 nodes respectively.

Table 5. The result of team 1

	2AS	4AS	6AS	8AS
Origin(no BW-DCache)	1.26ms	2.06ms	4.23ms	7.50ms
BW-DCache	2.16ms	2.42ms	2.74ms	3.89ms
	10AS	12AS	14AS	
Origin(no BW-DCache)	10.25ms	13.47ms	16.94ms	
BW-DCache	4.50ms	5.38ms	7.13ms	

Table 5 shows the test result. The fitting result is described in Fig.7. The read response time to the storage server of 10 clients without BW-DCache is 10ms; however, that of 18 clients with BW-DCache is 10.3ms. Therefore, the read response time of the two is approximately equal.

It should be noted that when the number of clients is two, our system is worse than the original. That is because that in this case, the storage server is not relatively busy; moreover, the response time of read from the storage server is smaller than that of read from the local BW-DCache. Consequently, the original system performs better than our system, when there are two clients.

**Fig. 7.** Comparison of response time

6.2.2 Effects of Cache Size

To study the effect of the cache size on our system performance, we varied the cache size. The cache size is set to 1.1G, 2.2G and 4.4G respectively. As mentioned above, due to limit of our experimental conditions, we only test the cases with 14 client nodes at the most. We adopt the least square method to fit the case with 16 nodes and 18 nodes respectively.

Table 6. The result of team 2

	2AS	4AS	6AS	8AS
1.1G	1.87 ms	2.69 ms	3.71 ms	4.87 ms
2.2G	2.03 ms	2.54 ms	3.29 ms	4.41 ms
4.4G	2.16 ms	2.42 ms	2.74 ms	3.89 ms
	10AS	12AS	14AS	
1.1G	6.18 ms	7.96 ms	10.18 ms	
2.2G	5.38 ms	6.74 ms	8.53 ms	
4.4G	4.50 ms	5.38 ms	7.13 ms	

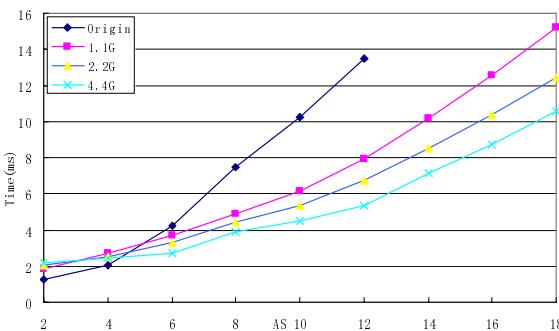
**Fig. 8.** Comparison of response time with different sizes

Table 6 shows the test result. We can see from Fig.8 that the read response time of 10 clients without BW-DCache is approximately equal to that of 14 clients with 1.1G cache size, 16 clients with 2.2G cache size, and 18 clients with 4.4G cache size, when maximal response time is limited to 10ms. Therefore, the number of clients which the server supports increases by 40%, 60% and 80%, when the cache size is set to 1.1G, 2.2G and 4.4G respectively.

6.2.3 Performance of Cache Management Strategy Optimization

The result is shown in Table 7. From the result, we can see that when the number of clients is small, the effect of cache management strategy optimization is relatively obvious. This is because when storage server is idle, the performance of reading from remote servers is better than that of reading from local BW-DCache. In this case, according to cache management strategy optimization mentioned in section 5.2, the system forwards the requests to remote servers directly to achieve better performance.

Table 7. The result of team 3

	2AS	4AS	6AS	8AS
Origin(no BW-DCache)	1.26ms	2.06ms	4.23ms	7.50ms
BW-DCache	2.16ms	2.42ms	2.74ms	3.89ms
Optimization	1.47ms	2.18 ms	2.57 ms	3.51 ms
	10AS	12AS	14AS	
Origin(no BW-DCache)	10.25ms	13.47ms	16.94ms	
BW-DCache	4.50ms	5.38ms	7.13ms	
Optimization	4.23ms	5.13ms	6.90ms	

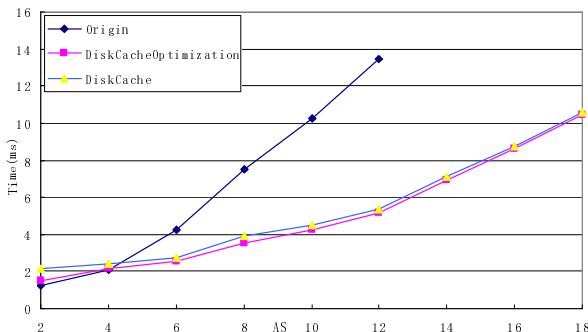


Fig. 9. Comparison of response time between management strategy optimization and original management strategy

However, with the increase of clients' number, the likelihood of server idle is reduced; therefore, at most of the time, the performance of reading from remote servers may be weaker than that of reading from local BW-DCache. In this case, the effect of cache management strategy optimization will be reduced. Similar to the case

mentioned above, due to limit of our experimental conditions, we only test the cases with 14 client nodes at the most. We adopt the least square method to fit the case with 16 nodes and 18 nodes respectively.

6.2.4 Performance of Replacement Algorithm Optimization

Table 8 shows the test result. As is mentioned in section 5.1, we modify the ARC algorithm by only employing more rigorous conditions, which don't introduce high overhead. From the result, comparing to original ARC, response time of replacement algorithm optimization is smaller. Moreover, the smaller is cache size, the more effective is the algorithm. This is because the performance impact introduced by algorithm distortion is larger, when cache size is relatively small. Similar to the case mentioned above, due to limit of our experimental conditions, we only test the cases with 14 client nodes at the most. We adopt the least square method to fit the case with 16 nodes and 18 nodes respectively.

Table 8. The result of team 4

	2AS	4AS	6AS	8AS
1G	1.79 ms	2.91 ms	4.11 ms	5.23 ms
1GOptimization	1.83 ms	2.72 ms	3.75 ms	4.91 ms
2G	1.92 ms	2.78 ms	3.51 ms	4.71 ms
2GOptimization	2.01 ms	2.61 ms	3.36 ms	4.58 ms
4G	2.04 ms	2.55 ms	2.91 ms	4.07 ms
4GOptimization	2.09 ms	2.49 ms	3.02 ms	4.01 ms
	10AS	12AS	14AS	
1G	6.62 ms	8.39 ms	11.01 ms	
1GOptimization	6.31 ms	8.01 ms	10.38 ms	
2G	5.87 ms	7.16 ms	9.03 ms	
2GOptimization	5.63 ms	6.98 ms	8.72 ms	
4G	4.86 ms	5.93 ms	7.56 ms	
4GOptimization	4.69 ms	5.62 ms	7.35 ms	

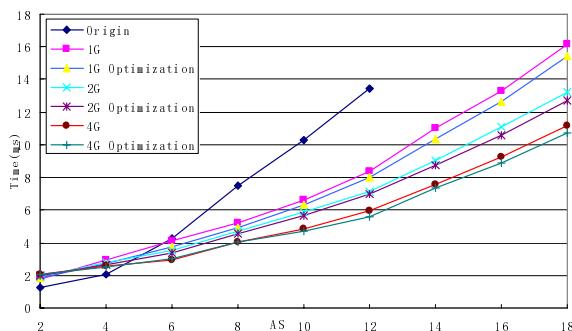


Fig. 10. Comparison of response time between replacement algorithm optimization and original replacement algorithm

7 Related Work

STICS[5] implemented shared cache system in networked storage environment, which could transfer scsi protocol into ip protocol. The system could not only connect to clients to provide private cache for clients, but also connect with storage device to provide shared cache for clients. xCachefs[6] provided a persistent cache frame. By the mechanism provided by the frame, any file system could buffer data into another file system with better performance; moreover, the frame kept consistency of cache and resource file system structure in order to provide file system service without restart after network disconnection between clients and servers. IBM Dm-cache[7] was a networked storage cache system which used local disk as cache. It provided semantic strategies based sessions to customize and manage cache system. Cooperative Caching [8][9] employed memory of other clients as the client cache to improve access performance of xFS. In AFS[2][3][4], a cache manager called Venus was implemented at the client to improve system performance. When a file was opened in a client, Venus duplicated the entire file from remote file server to local disk of the client. After that, read and write operations of the client are finished by employing local replicas. NFS[15][16](Network File System) adopted typical C/S mode, and communication between clients and servers is through RPC mechanism. NFS usually made use of client cache to improve system performance.

8 Conclusions and Future Work

In order to enhance QoS and scalability of BWFS, this paper analyses the bottleneck of scalability of BWFS in the non-linear editing application, and presents a solution called *BW-DCache*. In the solution, we first analyze the data access pattern of the application and employ local spare disk as a client cache to enhance system scalability by carefully choosing the cache algorithm and cache strategy. Furthermore, we implement the prototype system. Our experimental result shows that the scalability of BWFS is increased by 80% at the most. Moreover, we adopt some optimizations, including the replacement algorithm and cache strategy, to improve system performance further.

Though the experiment result illustrates the effectiveness of our solution, our data set is relative small due to some limit. In the future work, we will test large data set to verify our solution. In this paper, we adopt an existing cache algorithm. According to previous studies, it is more effective to design a dedicated algorithm for a special application. Our continuing work is to design a new replacement algorithm for the non-linear editing application.

Acknowledgements. This work was supported by Major State Basic Research Development Program of China (No.2004CB318205) and National High Technology Research and Development Program of China (No. 2007AA01Z402 and 2007AA01Z184).

References

1. Yang, D., Huang, H., Zhang, J., Xu, L.: BWFS: A Distributed File System with Large Capacity, High Throughput and High Scalability. *Journal of Computer Research and Development?* 42(6), 1028–1033 (2005)
2. Howard, J.H.: An Overview of the Andrew File System, CMU-ITC-062, <http://reports-archive.adm.cs.cmu.edu/itc85.html>
3. Tanenbaum, A.S.: Modern Operating Systems. Prentice-Hall, Englewood Cliffs (1992)
4. Howard, J.H., Kazar, M.L., et al.: Scale and performance in distributed file system. *ACM Transactions on Computer Systems* 6(1), 51–81 (1988)
5. He, X., Zhang, M., Yang, Q.: STICS: SCSI-to-IP cache for storage area networks. *Parallel and Distributed Computing* 64(9), 1069–1085 (2004)
6. Sivathanu, G., Zadok, E.: A Versatile Persistent Caching Framework for File Systems, Stony Brook University, Technical Report FSL-05-05 (2005)
7. Hensbergen, E.V., Zhao, M.: Dynamic policy disk caching for storage networking. Technical Report RC24123, IBM Research Division Austin Research Laboratory (2006)
8. Dahlin, M., et al.: Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In: Proceeding of the First Symposium on Operating Systems Design and Implementation, pp. 267–280 (1994)
9. Sarkar, P., Hartman, J.H.: Hint-based cooperative caching. *ACM Transactions on Computer Systems* 18(4), 387–419 (2000)
10. Denning, P.J., Schwartz, S.C.: Properties of the working set model. *Communications of the ACM* 15(3) (1972)
11. Smith, A.J.: Cache Memories. *ACM Computing Surveys* 14(3), 473–530 (1982)
12. Zhou, Y., Philbin, J.F., Li, K.: The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. In: Proc. of 2001 Annual USENIX Technical Conference (2001)
13. Johnson, T., Shasha, D.: 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In: Proceedings of the 20th International Conference on VLDB, pp. 439–450 (1994)
14. Megiddo, N., Modha, D.S.: ARC: A self-tuning, low overhead replacement cache. In: Proc. 2nd USENIX Conference on File and Storage Technologies (FAST 2003), San Francisco, CA, pp. 115–130 (2003)
15. Sun Microsystems, Inc. NFS: Network File System Protocol Specification (1989)
16. Callaghan, B., et al.: Sun Microsystems, Inc. NFS Version 3 Protocol Specification (1995)

Object-Oriented OpenMP Programming with C++ and Fortran

Christian Terboven, Dieter an Mey, Paul Kapinos, Christopher Schleiden,
and Igor Merkulow

JARA, RWTH Aachen University, Center for Computing and Communication
Seffenter Weg 23, 52074 Aachen, Germany
`{terboven,anmey,kapinos,schleiden,merkulow}@rz.rwth-aachen.de`

Abstract. Object-oriented programming has found its place in mainstream software application development since the 1990s and most current programming languages are either object-oriented by design, or have been retrofitted with such features. Given the recent trend towards multi-core processors, parallel programming is just about to become a mainstream requirement. In this work, we explore and compare techniques with which to parallelize object-oriented C++ and Fortran codes from the scientific domain with OpenMP. Our goal is to achieve the best possible performance while still maintaining the object-oriented programming style.

1 Introduction

The object-oriented (OO) programming paradigm employs techniques, such as encapsulation and modularity, to foster software development and maintenance. Encapsulation means hiding the implementation details of a software component behind a defined interface so that a change in one software component does not affect other components making use of it. Modularity separates concerns by enforcing logical boundaries among the components. Object-oriented programming has been in common use since the 1990s.

The compute-intense core of many partial differential equation solvers consists of Krylov subspace methods for iteratively solving linear equation systems with sparse matrices, for instance the generalized minimum residual (GMRES) method. Several C++ development projects were focused on how to implement such methods in an efficient manner using an OO approach, this includes the Navier-Stokes solver DROPS [7]. Fortran 95/2003 also provides virtually all of the necessary features for object-oriented design and implementation [4].

In this work, we explore and compare techniques to parallelize sparse linear-equation solvers and selected compute kernels with an OO programming style using OpenMP 3.0. Our primary goal is to maximize performance and scalability with the least impact on the elegance of the OO code; we want to preserve its aforementioned benefits. As developing and improving Krylov subspace solvers is an active field of numerical analysis, an approach that allows easy parallelization on today's ubiquitous multi-core architectures would be welcome. Furthermore,

we take OO abstractions to be an opportunity to introduce parallelism without hindering further program development: HPC specialists working on the parallelization and domain experts working on the algorithms can stick to the interfaces and, thus, escape creating diverging code versions. As has been explored before [6], common interfaces can also be used to experiment with different parallelization techniques so as to hide complex architectural tuning details from the application or algorithm developer.

The rest of this work is organized as follows: Section 2 presents the computational task that we studied, namely the parallelization of a CG-style iterative solver, and explains a few important concepts of OpenMP that have an impact on the implementation and performance. Section 3 discusses our parallelization strategies. In Section 4 we compare C++ and Fortran to assess how well the OO constructs of both programming languages work together with OpenMP parallelization, and we compare serial performance as well as scalability. We draw our conclusions and discuss future work in Section 5.

2 Computational Task and OpenMP

The excerpt in code 1 is taken from the C++ Navier-Stokes solver DROPS [2]. The abstractions of a matrix (class `MatrixCL`) and a vector (class `VectorCL`) employed in the code allow us to write numerical algorithms closely resembling the notation found in mathematical text books. The expression in line 7 implements a sparse matrix-vector-multiplication (SMXV), a dot-product can be found in line 8, lines 9 and 10 show vector operations. Given this code, well-suited to permit algorithmic modifications with little effort, our goal was to introduce parallelization with as little overhead as possible, while not breaking the OO programming style nor hindering ongoing development.

Although such an abstract notation is used in some C++ production codes already, for example in DROPS, in Fortran codes, a loop-oriented coding style is still dominating. The excerpt in Code 2 shows the same part of a GMRES-style

Code 1. Iteration loop of a CG-type solver in C++.

```

1 MatrixCL A(rows, cols, nonzeros);
2 VectorCL q(n), p(n), r(n);
3 [...]
4 for (int i = 1; i <= max_iter; ++i)
5 {
6     [...]
7     q = A * p;
8     double alpha = rho / (p*q);
9     x += alpha * p;
10    r -= alpha * q;
11    [...]

```

Code 2. Iteration loop of a CG-type solver in Fortran.

```

1 integer *8 nz_num
2 double precision a(nz_num), irow(n+1), icol(nz_num)
3 double precision x(n), r(n), p(n), q(n), alpha
4 [...]
5 do iter=1, max_iter
6     [...]
7     do i = 1, n
8         q(i) = 0.0d0
9         do j=irow(i), irow(i+1)-1
10            q(i)=q(i)+a(j)*p(icol(j))
11        end do
12    end do
13    alpha = 0.0d0
14    do i = 1, n; alpha = alpha + p(i) * q(i); end do
15    alpha = alpha / rho
16    do i = 1, n; x(i)= x(i) + alpha * p(i); end do
17    do i = 1, n; r(i) = r(i)-alpha * q(i); end do
18    [...]

```

solver as Code 1, this time in Fortran. The sparse matrix-vector-multiplication ranges from line 7 to 12, the dot-product ranges from line 13 to 15 and the lines 16 and 17 show the vector operations.

Parallelization of this code using an OpenMP **DO** worksharing construct would be straight forward and efficient, but to profit from an OO design, we wanted to introduce object-oriented abstractions to hide the parallelization in Fortran as well.

The OpenMP standard [1] supports the programming languages C, C++ and Fortran and allows for explicit parallelization via a so-called *Parallel Region*; it consists of a pragma and a structured block that will be transformed into parallel code by an OpenMP-aware compiler. The worksharing constructs of OpenMP distribute the work onto the threads of a parallel region; the most prominent worksharing construct is the **FOR** / **DO** construct to parallelize the execution of a loop. Note that each worksharing construct has an implicit barrier attached to its end; the barrier is a synchronization point after which the execution continues only when all threads have reached it. Obviously the end of a parallel region constitutes a synchronization point as well, because afterwards program execution continues with one thread only. This concept allows an incremental parallelization of an existing program, an important feature of OpenMP.

Although the OpenMP specification is defined so that all threads, except for the initial one, could be terminated after the execution of a parallel region and re-created when the next parallel region is encountered, most OpenMP implementations, such as the ones from Intel and Sun considered in our performance

comparison, make use of a *Thread Pool*. Using a Thread Pool, the threads are kept alive; after a parallel region they are only put to sleep and are instantly available at the next parallel region. The Thread Pool significantly reduces the overhead when multiple parallel regions are executed consecutively.

3 Parallelization Strategies for Object-Oriented Codes

Throughout this section, we discuss several strategies to parallelize object-oriented codes in C++ and Fortran. Given the recent trend towards cc-NUMA architectures, and its impact on the achievable performance, we also discuss a data model approach to take it into account.

3.1 Loop-Level Parallelization, C++

An obvious option for parallelizing the code presented in Code 1 is to factor it into a loop-style form to apply the OpenMP FOR worksharing constructs. With this approach, the code would become very similar to the initial Fortran code or C code. The factoring approach obviously breaks the object-oriented paradigm, and, therefore, we do not consider it to be a valid final result; nevertheless, we created it for performance comparison, and we will refer to it as *C++ (loop)* in Section 4.

3.2 Loop-Level Parallelization, Fortran

The initial Fortran code version is ready for the OpenMP FOR worksharing constructs: no code changes are necessary. In Section 4, we will refer to this version as *Fortran (loops)*. Besides of not enjoying the advantages of an OO code design, loop parallelization also comes with the disadvantage that it must be updated for all code changes made during the program's development.

To improve the efficiency of the *Fortran (loops)* approach, one can consider merging consecutive parallel regions (e.g., to save the overhead of taking threads in and out of the thread pool). Any performance increase comes at a high risk of introducing errors - such as data races - whenever a change is made in the code.

3.3 Parallelization via WORKSHARE, Fortran

The Fortran 90 language revision introduced array syntax allowing operations on arrays without explicit loops; the change greatly enhanced the readability of the code and can be considered as a basic OO abstraction. To support the parallelization of such constructs, OpenMP offers the WORKSHARE construct. Using this construct, the iteration loop of the GMRES-style code excerpt looks as shown in Code 3. The dot-product, for example, is found in line 10 and lines 9 and 11 are the associated OpenMP construct. We will refer to this approach as *Fortran (workshare)* in Section 4.

Like the loop-level parallelization, it is possible to merge consecutive parallel regions to save some OpenMP overhead.

Code 3. Iteration loop of a CG-type solver in Fortran using the WORKSHARE parallelization construct of OpenMP.

```

1 do iter=1, max_iter
2   [...]
3 !$omp parallel workshare
4   forall (i = 1:n)
5     q(i) = dot_product(a(irow(i):(irow(i+1)-1)),
6                         p(icol(irow(i):(irow(i+1)-1))))
7   end forall
8 !$omp end parallel workshare
9 !$omp parallel workshare
10  alpha = alpha / dot_product(p,q)
11 !$omp end parallel workshare
12 !$omp parallel workshare
13  x = x + alpha * p
14  r = r -alpha * q
15 !$omp end parallel workshare
16  [...]

```

3.4 Parallel Matrix and Vector Class, C++

As has been discussed earlier, maintaining the OO advantages is not possible with loop-level restructuring of the code; therefore, the obvious places to introduce the parallelization are the operator calls of the matrix and vector classes. To make this approach not only safe to use but also robust in case of code changes, each operator call has to host a self-contained parallel region. We discussed this approach for several different parallelization paradigms in [6], and two possible problems arise:

1. *Temporaries in compound expressions*: In case of a compound expression, such as $x = (a * 2.0) + b$; where x , a and b are vector data types, the compiler is allowed to introduce temporaries. Temporaries not only impose some sequential overhead but also can drastically decrease performance on cc-NUMA architectures.
2. *Overhead in compound expressions*: Because each operator call contains a complete parallel region, a compound expression such as $x = (a * 2.0) + b$; could lead to up to five parallel regions in this single expression: The computation of $a * 2.0$ and the assignment of the result to a temporary $_t1$, the computation of $_t1 + b$ and the assignment of the result to a temporary $_t2$, and, finally, the assignment of $_t2$ to x .

To overcome both problems of the C++ code, we developed an Expression Template [8] for our matrix and vector classes. Using expression templates, the right hand side of any compound expression is evaluated only by the assignment

operator. This avoids the creation of temporaries and allows for an efficient parallelization with OpenMP's worksharing constructs. Using this approach, the parallelization can be hidden completely from the application or algorithm developer. We refer to this version as *C++ (datatype)* in Section 4.

3.5 Parallel Matrix and Vector Data Type, Fortran

Although the array syntax already provides abstraction that can well be applied to parts of the linear algebra kernel, they still have to be considered basic. For example the sparse matrix-vector-multiplication cannot be hidden completely. To overcome this limitation, we introduced matrix and vector classes and defined appropriate operators to encapsulate the parallelization. Using native arrays as the vector representation is not possible because Fortran 95 does not permit one to override the pre-defined operators for intrinsic data types.

To represent the array needed to hold the data of a vector or a sparse matrix, respectively, we had to use **ALLOCATABLE** arrays. Fortran pointers would be flexible, but they turned out to be slower. Code 4 shows the parallel implementation of the assignment operator for our vector data type. The assignment operator, which would be provided by the compiler, would be serial.

Code 4. Parallel assignment operator for the vector data type in Fortran.

```

1 SUBROUTINE assign_VV ( left , right )
2   TYPE( vector ),INTENT(OUT) :: left
3   TYPE( vector ),INTENT(IN) :: right
4   INTEGER :: i
5   IF (.NOT.ALLOCATED( left%v ))
6     ALLOCATE( left%v( SIZE( right%v ) ) )
7 !$omp parallel do schedule(runtime)
8   do i = 1, SIZE( right%v ); left%v(i)=right%v(i); end do
9 !$omp end parallel do
10 END SUBROUTINE assign_VV

```

The problem of temporaries exists in Fortran as well. For each operator call, a check is made to assess whether the left hand side is already allocated, and, in case it is not, it has to be allocated explicitly as done in lines 5 and 6. Hence, a compound operation such as $x = (a * 2.0) + b$ must consist of multiple parallel regions, which involves some overhead.

Inside the operator implementation, we had to choose between array syntax or loop-based implementations, and for reasons discussed in Section 4, we decided to use a loop-based approach using OpenMP's DO worksharing construct. We refer to this version as *Fortran (datatype + loops)* in Section 4.

3.6 Handling cc-NUMA Architectures, C++ and Fortran

At present, all recent x86-based multi-socket systems implement a cache-coherent non-uniform memory architecture (cc-NUMA). To exploit the full performance of such machines, it is important to place the application's data in the system's memory carefully, because accessing remote memory leads to higher latency and lower bandwidth.

Like most current operating systems, Linux applies the so-called *First-Touch Strategy* [5] by physically placing a memory page on (or closest to) the processor that touches it first, typically during the initialization (with zero, for example). Once a page has been placed, it typically cannot be moved to a different location.

The C++ Standard Template Library (STL) provides an allocator to encapsulate memory management. We depended on that concept to bring cc-NUMA optimization into our application by providing two allocators:

1. `dist_allocator`: The data are distributed according to a specified OpenMP schedule. It is implemented via an OpenMP `FOR` loop construct initializing the data with zero with the intent that the same scheduling will be used in the actual computation phase later on.
2. `chunked_allocator`: The data are distributed according to a user-defined scheme. This is implemented using explicit initialization according to the mapping of array slices to NUMA nodes. For instance, one can partition a sparse matrix into as many slices as there are threads, all consisting of approximately the same number of nonzeros. The idea is to minimize remote accesses as well as to achieve a near-optimal load balance.

In the Fortran language, there is no construct like the allocator, but one can achieve the same functionality, of course. Because Fortran does not support class constructors, the functionality has to be implemented in an initialization function that is called whenever a matrix or vector data type is instantiated.

For both programming languages, the user has to take care of thread binding manually; otherwise the cc-NUMA optimization has no effect if the operating system moves a thread to a different NUMA node. For our performance measurements, we used the approach named `dist_allocator`, because it does not depend on a given input data set.

4 Comparison: C++ versus Fortran

In this Section we compare the parallelization strategies discussed in Section 3 regarding programmability and performance.

4.1 Programmability

Regardless of whether one uses C++ or Fortran, a purely loop-style parallelization has several disadvantages compared to an object-oriented approach. Fortran's array syntax makes a small step in the direction of abstraction, but we

achieved the desired OO design only by introducing custom data types to represent a matrix and a vector. This holds for C++ as well.

Given our data type design, we achieved the parallelization of the computational task by placing self-contained parallel regions inside operators calls. As a result, we gained a significant advantage: the parallelization itself is completely invisible to the application or algorithm developer, respectively. Because of the internal parallelization, the data types are safe to use, and code changes do not lead to data races, or like errors.

In C++ we were able to use an Expression Template framework to parallelize compound expressions in a single parallel region, thereby avoiding temporaries, which reduces the overhead of the parallelization itself as will be discussed in the following subsection. The same strategy is not possible in Fortran, but this has no influence of the programmability. In summary, we are now able to write numerical algorithms resembling mathematical text book notation, as was our stated goal.

Code 5 shows the parallel version of the CG-type solver in Fortran using our data types and the internal parallelization with OpenMP. In the C++ code, there is no visible difference between the serial and the parallel version as well, because we had only to change declarations via a `typedef` to make use of our parallelized data types.

Code 5. Iteration loop of the *parallel* CG-type solver in Fortran with parallel object-oriented data types.

```

1 USE matvect
2 TYPE(matcrs):: a
3 TYPE(vector) :: x, r, p, q
4 DOUBLE PRECISION :: alpha
5 [...]
6 do iter=1, max_iter
7   [...]
8   q = a * p
9   alpha = norm_2(p,q)
10  x = x + alpha * p
11  r = r -alpha * q
12  [...]
```

4.2 Performance

All performance measurements discussed here were carried out on Sun Lynx blade systems equipped with two Intel Xeon X5570 processors (Nehalem) with a clock speed of 2.93 GHz running Linux. We examined both the Intel C/C++/Fortran 11.0 compiler suite and the Sun Fortran compiler 8.3; the Sun C++ compiler could not compile our C++ code correctly.

DAXPY-operation from STREAM. To evaluate our new techniques, we looked first at the performance of STREAM-like [3] operations, because memory bandwidth typically is a critical issue for sparse solvers. The DAXPY part of that benchmark is of special interest, because it represents a compound expression. Figure 1 shows the memory bandwidth [MB/s] with up to 16 threads for the loop versions in C++ and Fortran and the data type version in C++, all using the Intel 11.0 compiler. All arrays have a dimension of 300 MB.

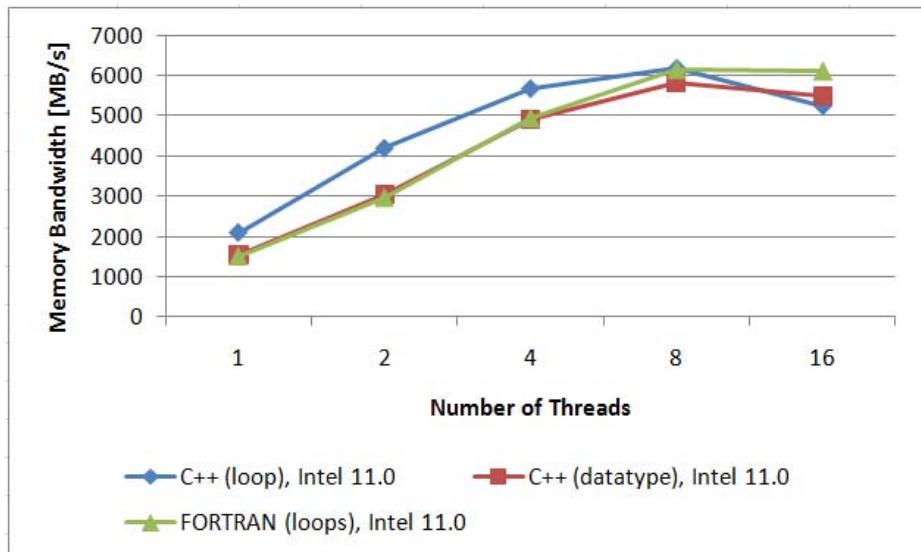


Fig. 1. Selected results of DAXPY kernel, part 1

For up to four threads, the C++ loop version outperforms both the Fortran loop version and the C++ data type version. Because we used allocatable arrays, the Fortran compiler was not able to carry out all optimizations, while in the C++ loop version, we were able to use the `restrict` keyword to aid the compiler in optimizing the code; therefore, the C++ loop version is the fastest of the three. Using eight threads, all versions deliver approximately the same performance, because the memory bandwidth was exhausted.

Although the behaviour of these kernels in relation to each other was what we expected, we did not achieve the absolute performance that we expected for a STREAM-type kernel on that architecture, and are still investigating why.

As can be seen in figure 2, the workshare version, using the Intel compiler, and the data type versions did not perform well, while the workshare version using the Sun compiler delivers about the same performance as the C++ loop version using the Intel compiler. The reason for the poor performance of the Intel workshare version is that the current version of the Intel compiler does not parallelize OpenMP WORKSHARE constructs.

The data type versions, independent of the compiler, start with a significantly slower serial performance and show only modest speed up. The difficulty is high overhead, because the DAXPY operation involves 3 parallel regions in the Fortran data type version, whereas it involves only one in the C++ data type version, because of the Expression Template framework.

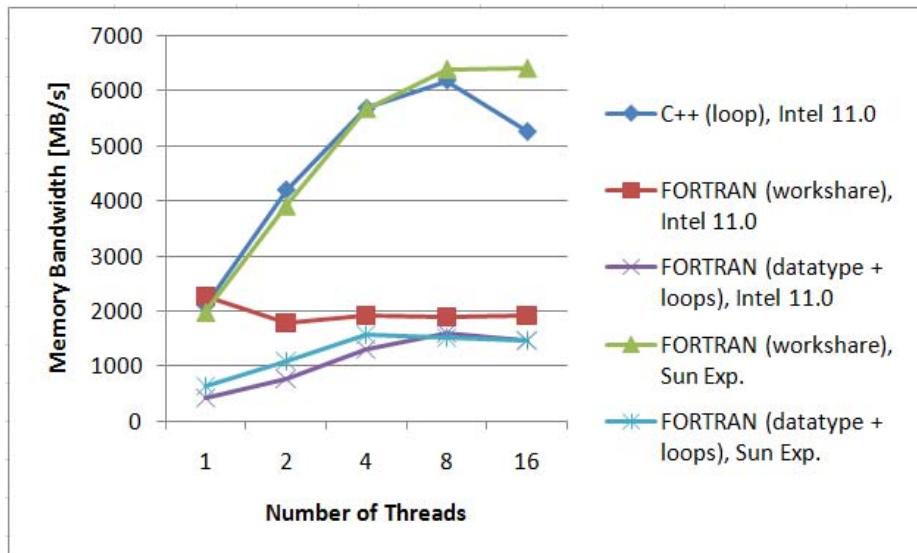


Fig. 2. Selected results of DAXPY kernel, part 2

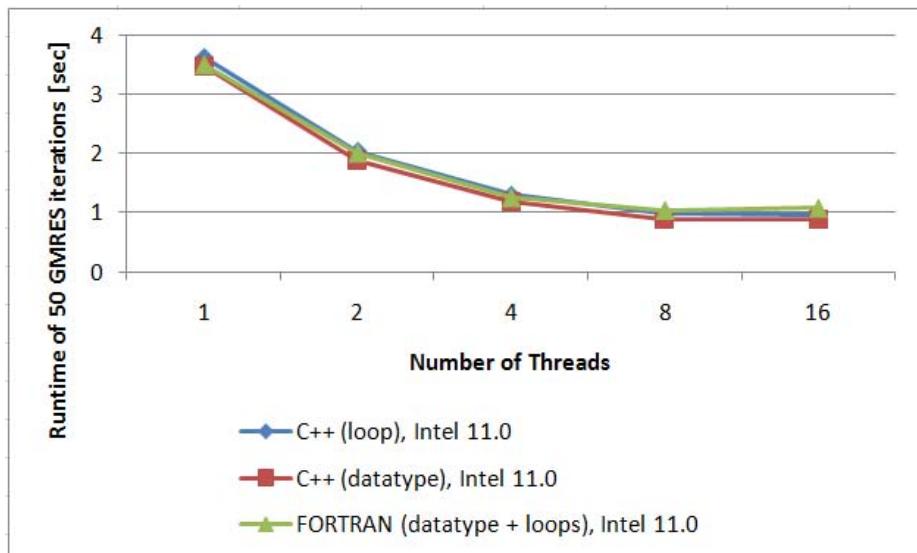


Fig. 3. Selected results of MGMRES iterative solver, part 1

MGMRES Iterative Solver. We also examined a GMRES solver applied to a linear equation system that comes in the two-phase flow simulation package DROPS [2]. Figure 3 shows the runtime comparison [sec] of the MGMRES solver for 50 iterations on a matrix with approximately 20,000,000 non-zero elements. As can clearly be seen, the performance difference is very small, as all versions reside in the same range of scalability. The C++ data type version is the fastest, because of the efficient implementation of compound expressions. Fortran’s disadvantage in that respect is not of great significance here, because the runtime of that solver is dominated by the sparse matrix-vector-multiplication; it can also be implemented efficiently with Fortran.

In summary, we conclude that, for this algorithm, the OO abstractions are for free in terms of runtime costs when implemented correctly.

5 Conclusion and Future Work

In this paper, we showed how object-oriented abstractions can be exploited to hide parallelization and its details from the application or algorithm developer, respectively. In C++, we used an Expression Template framework to implement our approach very efficiently, but with a significant amount of coding effort. We achieved the same programmability in Fortran, but because compound expressions cannot be parallelized with a single parallel region, the overhead is higher than for C++. Our approach of using self-contained OpenMP parallel regions inside operator calls turned out to be very elegant and safe.

We are currently working to eliminate the implicit barriers of the OpenMP worksharing constructs. To do so, we aim to have the parallel region spanning the whole solver and to use orphaned worksharing constructs inside the operator calls. If the distribution of work onto threads can be taken into account in the data type operations, most barriers should be eliminated while still providing robustness against code changes in terms of correctness.

Finally, our study shows that Fortran is still missing functionality in regard to support for OpenMP parallelization of object-oriented codes; OpenMP has no explicit support for Fortran 2003 yet. Currently, we are working to identify language features that could be supported in the next version of OpenMP.

References

1. ARB. OpenMP Application Program Interface, version 3.0 (May 2008)
2. Gross, S., Peters, J., Reichelt, V., Reusken, A.: The DROPS Package for Numerical Simulations of Incompressible Flows using Parallel Adaptive Multigrid Techniques. IGPM-Report, vol. 211 (2002)
3. McCalpin, J.: Stream: Sustainable memory bandwidth in high performance computers (1999)
4. Metcalf, M., Reid, J., Cohen, M.: Fortran 95/2003 explained. Oxford University Press, Oxford (2004)

5. Terboven, C., an Mey, D., Schmidl, D., Jin, H., Reichstein, T.: Data and Thread Affinity in OpenMP Programs. In: Workshop Memory Access on future Processors: A solved problem?, ACM International Conference on Computing Frontiers, Ischia, Italy (May 2008)
6. Terboven, C., Schleiden, C., an Mey, D.: Comparing Programmability and Scalability of Multicore Parallelization Paradigms with C++. In: Ayguade, E., Gioiosa, R., Stenstrom, P., Unsal, O. (eds.) Second Workshop on Programmability Issues for Multi-Core Computers (MULTIPROG-2), Paphos, Cypress (January 2009)
7. Terboven, C., Spiegel, A., an Mey, D., Gross, S., Reichelt, V.: Parallelization of the C++ Navier-Stokes Solver DROPS with OpenMP. In: Joubert, G.R., Nagel, W.E., Peters, F.J., Plata, O., Tirado, P., Zapata, E. (eds.) Parallel Computing (ParCo 2005): Current & Future Issues of High-End Computing, Malaga, Spain, September 2005. NIC Series, vol. 33, pp. 431–438 (2005)
8. Veldhuizen, T.: Expression templates. C++ Report 7, 26–31 (1995)

FFT-Based Dense Polynomial Arithmetic on Multi-cores

Marc Moreno Maza¹ and Yuzhen Xie²

¹ Ontario Research Centre for Computer Algebra
University of Western Ontario, London, Canada
`moreno@csd.uwo.ca`

² Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge, USA
`yxie@csail.mit.edu`

Abstract. We report efficient implementation techniques for FFT-based dense multivariate polynomial arithmetic over finite fields, targeting multi-cores. We have extended a preliminary study dedicated to polynomial multiplication and obtained a complete set of efficient parallel routines in Cilk++ for polynomial arithmetic such as normal form computation. Since bivariate multiplication applied to balanced data is a good kernel for these routines, we provide an in-depth study on the performance and the cut-off criteria of our different implementations for this operation. We also show that, not only optimized parallel multiplication can improve the performance of higher-level algorithms such as normal form computation but also this composition is necessary for parallel normal form computation to reach peak performance on a variety of problems that we have tested.

Keywords: Parallel polynomial arithmetic, parallel polynomial multiplication, parallel normal form, parallel multi-dimensional FFT/TFT, Cilk++, multi-core.

1 Introduction

Polynomial Arithmetic is at the core of every computer algebra system (CAS) such as AXIOM, MAGMA, MAPLE, MATHEMATICA, NTL and REDUCE, and has an essential impact on the performance of these software packages. Today, the ubiquity of hardware acceleration technologies (multi-cores, graphics processing units, ...) makes the development of *basic polynomial algebra subroutines (BPAS)* necessary in order to support CAS, akin to the BLAS in numerical linear algebra.

The work presented in this paper aims at contributing to this effort. In fact, and up to our knowledge, this is the first report on the parallelization of dense polynomial arithmetic, over finite fields and targeting multi-cores. All symbolic calculations on univariate and multivariate polynomials can be reduced to computing with polynomials over finite fields (such as the prime field Z/pZ for a prime number p) via the so-called *modular techniques*. Moreover, most symbolic calculations tend to densify intermediate expressions even when the input and output polynomials are sparse. See Chapter 5 in [10] for an extensive presentation of these ideas, which explain why we focus primarily on dense polynomials over finite fields.

Such polynomials are well suited for the use of asymptotically fast algorithms based on FFT techniques. Note that some features of FFT techniques are specific to finite

fields, see Section 2.1 for details. In this context polynomial multiplication plays a central role, and many basic operations on polynomials such as division can be efficiently reduced to multiplication. This observation has motivated our preliminary study [19] dedicated to FFT-based dense polynomial multiplication. We have shown that *balanced input data* can maximize parallel speedup and minimize cache complexity for bivariate multiplication. We say that a pair of multivariate polynomials is *balanced* if the partial degrees of their product are equal (or very close). However, unbalanced input data, which are common in symbolic computation, are challenging. We have provided efficient techniques to reduce multivariate (and univariate) multiplication to *balanced bivariate multiplication*. Our implementation in Cilk++ [3] demonstrates good speedup on multi-cores. Sections 2.2 to 2.4 summarize the context and the results of [19].

In order to obtain a solid foundation library for basic polynomial algebra subroutines over finite fields and targeting multi-cores, at least two essential problems need to be handled and are addressed in this paper. In Sections 1.1 and 1.2, we describe these problems and present our solutions, which are detailed through Sections 3 to 6. Section 1.3 describes our experimentation framework. Finally, in Section 7 we summarize our results and discuss the outcome of this research.

1.1 Optimizing Balanced Bivariate Multiplication

FFT-based bivariate multiplication can be achieved by a variety of algorithms and implementation techniques. Since balanced bivariate multiplication is the kernel to which we are reducing multivariate multiplication, we need to determine the most appropriate algorithm and implementation techniques for the input patterns of practical interest. This problem of *cut-off criteria* is essential in scientific computing. For instance, in coding matrix multiplication, one is faced with choosing among Strassen multiplication, classical multiplication and others; see [12] for details.

Determining cut-off criteria is even more challenging in the context of multi-core programming where both input data patterns and number of cores need to be taken into account. This makes it necessary to combine theoretical and empirical analysis. The former cannot provide precise criteria due to simplification hypotheses but helps narrowing the pattern ranges of the latter. Once the empirical results are obtained, the theoretical analysis can also help understanding them.

In this work, we consider two implementations of bivariate multiplication. One is based on Cooley-Tukey FFT; often we simply call it FFT. The other is based on Truncated Fourier Transform (TFT), see Sections 2.3. The theoretical analysis provides a simple cut-off criterion between our two algorithms when run serially. Section 4, after a description of our implementation, provides experimental results on 1, 8, 12 and 16 cores on a 16-core machine. We obtain simple cut-off criteria in each case and for several degree patterns of practical interest. These results are important since for certain degree ranges and for certain numbers of cores TFT substantially outperforms FFT while FFT is faster in the other cases. Taking advantage of these features can speedup not only multiplication but also the operations that rely on it.

1.2 Efficient Parallel Computation of Normal Forms

All basic operations on polynomials, such as division, can be reduced to multiplication. For multivariate polynomials over finite fields, two basic operations are of high interest: *exact division* and *normal form computations*. Note that polynomial addition is important too but does not bring any particular implementation challenges.

Given two multivariate polynomials f and g , we say that g divides f exactly if there exists a polynomial q such that $f = qg$ holds. Testing whether g divides f exactly and computing q when this holds can be done using FFT-techniques similar to those used for multiplication in Section 2.3. Hence we do not insist on this operation since no major additional implementation issues have to be handled with respect to multiplication.

However, computing normal forms (as defined in Section 2.5) brings new challenges. Indeed, complexity estimates show that the serial and parallel times of these computations (using the multi-threaded programming model of [9]) are exponential in the number of variables, see Section 5. Moreover, the number of synchronization points in a parallel program computing normal forms is also exponential in the number of variables. In addition, at each synchronization point the number of threads which need to join grows with the input data size (precisely with their partial degrees). Consequently, the parallel overhead is potentially large. A first attempt for parallelizing the serial normal form algorithms of [17] reached limited success as reported in [14].

In this work, we investigate how our parallel multiplication code could be efficiently composed with a parallel normal form implementation. This has the potential to increase parallel speedup factors but also parallel overhead.

Once again we approach this problem by combining theoretical and empirical analysis. In the former, some parallel overhead (the one for parallelizing a `for` loop) is taken into account, but not all. For instance, those coming from synchronization points are neglected. In Section 5, this theoretical analysis suggests that parallel multiplication can improve the parallelism of parallel normal form computation. In Section 6, our experimentation not only confirms this insight but shows that parallel multiplication is necessary for parallel normal form computation to reach good speedup factors on all input patterns that we have tested. These results are important since normal form computations represent the dominant cost in many higher-level algorithms, such as those for solving systems of polynomial equations, which is our driving application.

1.3 Experimentation Framework

The techniques proposed in this paper are implemented in the Cilk++ language [3], which extends C++ to the realm of multi-core programming based on the multi-threaded model realized in [9]. The Cilk++ language is also equipped with a provably efficient parallel scheduler by work-stealing [2]. We use the serial C routines for 1-D FFT and 1-D TFT from the modpn library [16]. Our integer arithmetic modulo a prime number relies also on the efficient functions from modpn, in particular the improved Montgomery trick [18], presented in [17]. All our benchmarks are carried out on a 16-core machine with 16 GB memory and 4096 KB L2 cache. All the processors are Intel Xeon E7340 @ 2.40GHz.

2 Background

Throughout this paper \mathbb{K} designates the finite field Z/pZ with p elements, where $p > 2$ is a prime number. All polynomials considered hereafter are multivariate with coefficients in \mathbb{K} and with n ordered variables $x_1 < \dots < x_n$. The set of all such polynomials is denoted by $\mathbb{K}[x_1, \dots, x_n]$.

The purpose of this section is to describe algorithms for two basic operations in $\mathbb{K}[x_1, \dots, x_n]$: *multiplication* and *normal form computation*. These algorithms are based on FFT techniques. We start by stressing the specificities of performing FFTs over finite fields, in particular the use of the *Truncated Fourier Transform* (TFT). In the context of polynomial system solving, which is our driving application, this leads to what we call the *1-D FFT black box assumption*.

2.1 FFTs over Finite Fields and the Truncated Fourier Transform

Using the Cooley-Tukey algorithm [6] (and its extensions such as Bluestein's algorithm) one can compute the *Discrete Fourier Transform* (DFT) of a vector of s complex numbers within $O(s \lg(s))$ scalar operations. For vectors with coordinates in the prime field \mathbb{K} , three difficulties appear with respect to the complex case.

First, in the context of symbolic computation, it is desirable to restrict ourselves to radix 2 FFTs since the radix must be invertible in \mathbb{K} and one may want to keep the ability of computing modulo small primes p , even $p = 3, 5, 7, \dots$ for certain types of modular methods, such as those for polynomial factorization; see [10, Chapter 14] for details. As a consequence the FFT of a vector of size s over the finite field \mathbb{K} has the same running time for all s in a range of the form $[2^\ell, 2^{\ell+1})$. This *staircase* phenomenon can be smoothed by the so-called *Truncated Fourier Transform* (TFT) [11]. In most practical cases, the TFT performs better in terms of running time and memory consumption than the radix-2 Cooley-Tukey Algorithm; see the experimentation reported in [17]. However, the TFT has its own practical limitations. In particular, no efficient parallel algorithm is known for it.

Another difficulty with FFTs over finite fields comes from the following fact: a primitive s -th root of unity (which is needed for running the Cooley-Tukey algorithm on a vector of size s) exists in \mathbb{K} if and only if s divides $p - 1$. Consider two univariate polynomials f, g over \mathbb{K} and let d be the degree of the product fg . It follows that fg can be computed by evaluation and interpolation based on the radix 2 Cooley-Tukey Algorithm (see the algorithm of Section 2.3 with $n = 1$) if and only if some power of 2 greater than d divides $p - 1$. When this holds, computing fg amounts to:

- $\frac{9}{2} \lg(s)s + 3s$ operations in \mathbb{K} using the Cooley-Tukey Algorithm,
- $\frac{9}{2}(\lg(s) + 1)(d + 1) + 3s$ operations in \mathbb{K} using TFT,

where s is the smallest power of 2 greater than d . When this does not hold, one can use other techniques, such as the Schönage-Strassen Algorithm [10, Chapter 8], which introduces “virtual primitive roots of unity”. However, this increases the running time to $O(s \lg(s) \lg(\lg(s)))$ scalar operations.

Last but not least, when solving systems of algebraic equations (which is our driving application), partial degrees of multivariate polynomials (including degrees of univariate polynomials) rarely go beyond the million. This implies that, in our context, the

lengths of the vectors to which 1-D FFT need to be applied are generally not large enough for making efficient use of parallel code for 1-D FFT.

2.2 The 1-D FFT Black Box Assumption

The discussion of the previous section, in particular its last paragraph, suggests the following hypothesis. We assume throughout this paper that we have at our disposal a **black box** computing the DFT at a 2^ℓ -primitive root of unity (when \mathbb{K} admits such value) of any vector of size s in the range $(2^{\ell-1}, 2^\ell]$ in time $O(s \lg(s))$. However, we do not make any assumptions about the algorithm and its implementation. In particular, we do not assume that this implementation is a parallel one. Therefore, we rely on the row-column multi-dimensional FFT to create concurrent execution in the algorithm presented in Section 2.3.

2.3 Multivariate Multiplication

Let $f, g \in \mathbb{K}[x_1, \dots, x_n]$ be two multivariate polynomials. For each i , let d_i and d'_i be the degree in x_i of f and g respectively. For instance, if $f = x_1^3 x_2 + x_3 x_2^2 + x_3^2 x_1^2 + 1$ we have $d_1 = 3$ and $d_2 = d_3 = 2$. We assume the existence of a primitive s_i -th root ω_i , for all i , where s_i is a power of 2 satisfying $s_i \geq d_i + d'_i + 1$. Then, the product fg is computed as follows.

Step 1: Evaluate f and g at each point of the n -dimensional grid $((\omega_1^{e_1}, \dots, \omega_n^{e_n}), 0 \leq e_1 < s_1, \dots, 0 \leq e_n < s_n)$ via multi-dimensional FFT.

Step 2: Evaluate fg at each point P of the grid, simply by computing $f(P)g(P)$.

Step 3: Interpolate fg (from its values on the grid) via multi-dimensional FFT.

The above procedure amounts to:

$$\frac{9}{2} \sum_{i=1}^n \left(\prod_{j \neq i} s_j \right) s_i \lg(s_i) + (n+1)s = \frac{9}{2}s \lg(s) + (n+1)s \quad (1)$$

operations in \mathbb{K} , where $s = s_1 \cdots s_n$. If our 1-D FFT black box relies on TFT rather than the Cooley-Tukey algorithm, the above estimate becomes:

$$\frac{9}{2} \sum_{i=1}^n \left(\prod_{j \neq i} s_j \right) (d_i + d'_i + 1) (\lg(s_i) + 1) + (n+1) \prod_{i=1}^n (d_i + d'_i + 1). \quad (2)$$

2.4 Balanced Bivariate Multiplication

In [19], the authors give a cache complexity estimate of the algorithm of Section 2.3 under the assumption of 1-D FFT black box. Using the theoretical model introduced in [8], and denoting by L the size of a cache line, they have obtained the following upper bound $c s \frac{n+1}{L} + c s \left(\frac{1}{s_1} + \cdots + \frac{1}{s_n} \right)$ for some constant $c > 0$ on the number of cache misses. This suggests the following definition. The pair of polynomials f, g is said *balanced* if all the partial degrees of their product are equal, that is, if $d_1 + d'_1 = d_i + d'_i$

holds for all $2 \leq i \leq n$. Indeed, for fixed s and n , this bound is minimized when the pair f, g is balanced; moreover it reaches a local minimum at $n = 2$ and $s_1 = s_2 = \sqrt{s}$. Experimentation reported in [19] confirms the good performance of *balanced bivariate multiplication*, that is, bivariate multiplication with balanced input. Based on these results, the authors have developed techniques in order to efficiently reduce any dense multivariate polynomial multiplication to balanced bivariate multiplication.

2.5 Normal Form Computation

Let $f, g_1, \dots, g_n \in \mathbb{K}[x_1, \dots, x_n]$ be polynomials. Recall that variables are ordered as $x_1 < \dots < x_n$. We assume that the set $\{g_1, \dots, g_n\}$ satisfies the following properties:

- (i) for all $1 \leq i \leq n$ the polynomial g_i is non-constant and its largest variable is x_i ,
- (ii) for all $1 \leq i \leq n$ the leading coefficient of g_i w.r.t. x_i is 1,
- (iii) for all $2 \leq i \leq n$ and all $1 \leq j < i$ the degree of g_i in x_j is less than the degree of g_j in x_j , that is, $\deg(g_i, x_j) < \deg(g_j, x_j)$.

Such a set is called a *reduced monic triangular set*. The adjectives triangular, monic and reduced describe respectively the above properties (i), (ii) and (iii). We will denote by δ_i the degree of g_i in x_i and by δ the product $\delta_1 \cdots \delta_n$. For instance, with $n = 2$, $g_1 = x_1^2 + 1$ and $g_2 = x_2^3 + x_1$, the set $\{g_1, g_2\}$ is a reduced monic triangular set. Observe that this notion is dependent on the variable ordering. In our example, the set $\{g_1, g_2\}$ would no longer be triangular for the ordering $x_2 < x_1$.

Reduced monic triangular sets are special cases of Gröbner bases [7] and enjoy many algorithmic important properties. We are interested here in the following one. There exists a **unique** polynomial $r \in \mathbb{K}[x_1, \dots, x_n]$ such that the following hold:

- (iv) either $r = 0$ holds or for all $1 \leq i \leq n$ the degree of r in x_i is less than δ_i ,
- (v) f is congruent to r modulo $\{g_1, \dots, g_n\}$, that is, there exist polynomials $q_1, \dots, q_n \in \mathbb{K}[x_1, \dots, x_n]$ such that we have $f = r + q_1g_1 + \dots + q_ng_n$.

Such a polynomial r is called the *normal form* of f w.r.t. $\{g_1, \dots, g_n\}$. Consider n, g_1, g_2 as above and $f = x_2^3x_1 + x_2x_1^2$. Then $r = -x_2 + 1$ is the normal form of f w.r.t. $\{g_1, g_2\}$. Indeed we have $f = r + q_1g_1 + q_2g_2$ with $q_2 = x_1$ and $q_1 = x_2 - 1$; moreover we have $\deg(r, x_1) < \delta_1$ and $\deg(r, x_2) < \delta_2$.

In broad terms the polynomial r is obtained after simplifying f w.r.t. $\{g_1, \dots, g_n\}$. It is, indeed, what the command `simplify` computes in computer algebra systems such as MAPLE, when this command is applied to f and $\{g_1, \dots, g_n\}$. This is an essential operation in symbolic computation and the above result states that r is uniquely defined as long as it satisfies (iv) and (v). One natural way for computing r is as follows:

- (a) Initialize r_{n+1} to be f .
- (b) For i successively equal to $n, n-1, \dots, 2, 1$ compute r_i as the remainder in the Euclidean division of r_{i+1} by g_i , regarding these polynomials as univariate in x_i .
- (c) Return r_1 .

In our example we set $r_3 = f$ and compute r_2 the remainder of r_3 by g_2 which is $r_2 = x_2x_1^2 - x_1^2$. Then we compute r_1 the remainder of r_2 by g_1 which is $-x_2 + 1$.

This procedure suffers from intermediate expression swell. (This cannot be seen on very simple example, of course.) One can check that the degree in x_1 of the successive remainders r_n, \dots, r_2 may dramatically increase until r_2 is finally divided by g_1 . Consider $g_n = x_n^2 - x_1^2 - 1, g_{n-1} = x_{n-1}^2 - x_1^2 - 1, \dots, g_2 = x_2^2 - x_1^2 - 1, g_1 = x_1^2 + 1$ and $f = x_2^4 \cdots x_{n-1}^4 x_n^4$. We will obtain $r_2 = (x_1^2 + 1)^{2n-2}$ whereas r is simply 0.

This phenomenon is better controlled by an algorithm proposed by Li, Moreno Maza and Schost in [17]. This latter procedure relies on Cook-Sieveking-Kung's "fast division trick" [520][13] which reduces an Euclidean division to two multiplications. The algorithm of [17] proceeds by induction on the number of variables and its pseudo-code is shown below. The base case, that is $n = 1$, is given by the procedure NormalForm_1 which is in fact Cook-Sieveking-Kung's fast division. The general case is given by the NormalForm_i procedure for $2 \leq i \leq n$ where the input polynomial f is assumed to be in $\mathbb{K}[x_1, \dots, x_i]$ and $\{g_1, \dots, g_i\}$ is a reduced monic triangular set in $\mathbb{K}[x_1, \dots, x_i]$. A few comments are needed about these two procedures.

- $\text{Rev}(g_i)$ designates the *reversal* of the polynomial g_i , that is, the polynomial obtained from g_i by reversing the order of its coefficients; for instance $\text{Rev}(g_1) = 3x_1^2 + 2x_1 + 1$ for $g_1 = x_1^2 + 2x_1 + 3$.
- $\text{Rev}(g_i)^{-1} \pmod{g_1, \dots, g_{i-1}, x_i^{\deg(f, x_i) - \deg(g_i, x_i) + 1}}$ is the inverse of $\text{Rev}(g_i)$ modulo the reduced monic triangular set $\{g_1, \dots, g_{i-1}, x_i^{\deg(f, x_i) - \deg(g_i, x_i) + 1}\}$; this can be computed via *symbolic Newton Iteration*, see [10] Chapter 8].
- In practice the quantities S_1, \dots, S_n are pre-computed and stored before calling NormalForm_i . Therefore, the computations of these quantities are not taken into account in any complexity analysis of these procedures.
- $\text{map}(\text{NormalForm}_{i-1}, \text{Coeffs}(f, x_i), \{g_1, \dots, g_{i-1}\})$ is the polynomial in x_i obtained from f by replacing each coefficient of f in x_i with its normal form w.r.t. $\{g_1, \dots, g_{i-1}\}$.

$\text{NormalForm}_1(f, \{g_1\})$

- 1 $S_1 := \text{Rev}(g_1)^{-1} \pmod{x_1^{\deg(f, x_1) - \deg(g_1, x_1) + 1}}$
- 2 $D := \text{Rev}(f)S_1 \pmod{x_1^{\deg(f, x_1) - \deg(g_1, x_1) + 1}}$
- 3 $D := g_1 \text{Rev}(D)$
- 4 **return** $f - D$

$\text{NormalForm}_i(f, \{g_1, \dots, g_i\})$

- 1 $f := \text{map}(\text{NormalForm}_{i-1}, \text{Coeffs}(f, x_i), \{g_1, \dots, g_{i-1}\})$
- 2 $S_i := \text{Rev}(g_i)^{-1} \pmod{g_1, \dots, g_{i-1}, x_i^{\deg(f, x_i) - \deg(g_i, x_i) + 1}}$
- 3 $D := \text{Rev}(f)S_i \pmod{x_i^{\deg(f, x_i) - \deg(g_i, x_i) + 1}}$
- 4 $D := \text{map}(\text{NormalForm}_{i-1}, \text{Coeffs}(D, x_i), \{g_1, \dots, g_{i-1}\})$
- 5 $D := g_i \text{Rev}(D)$
- 6 $D := \text{map}(\text{NormalForm}_{i-1}, \text{Coeffs}(D, x_i), \{g_1, \dots, g_{i-1}\})$
- 7 **return** $f - D$

Observe that performing NormalForm_1 simply amounts to two multiplications. More generally, NormalForm_i requires two multiplications and $3(\delta_i + 1)$ recursive calls to NormalForm_{i-1} . In [17], the authors have shown that $\text{NormalForm}_n(f, \{g_1, \dots, g_n\})$ runs in $O(4^n \delta \lg(\delta) \lg(\lg(\delta)))$ operations in \mathbb{K} .

3 Cutoff Analysis for Dense Bivariate Multiplication

As mentioned in the introduction, the work in [19] identified balanced bivariate multiplication as a good kernel for dense multivariate multiplication. Bivariate multiplication based on FFT techniques, and under the 1-D FFT black box assumption, can be done via either 2-D FFT or 2-D TFT. More precisely, the necessary 1-D FFTs of the algorithm of Section 2.3 can be performed via either the Cooley-Tukey algorithm or TFT. In order to optimize our balanced bivariate multiplication code, we need to determine when to use these different 1-D FFT routines. This section offers a first answer based on algebraic complexity analysis meanwhile Section 4 will provide experimental results.

Recall that we aim at applying bivariate multiplication to balanced pairs. In practice, as mentioned in [19], using “nearly balanced” pairs is often sufficient. Hence, with the notations of Section 2, we can assume that $d_1 + d'_1$ and $d_2 + d'_2$ are of the same order of magnitude. Moreover, it is often the case that d_i and d'_i are quite close, for all $1 \leq i \leq n$. For instance, in normal form computations, we have $d_i \leq 2d'_i - 2$ for all $1 \leq i \leq n$ (up to exchanging the role of f and g). Therefore, in both the experimental analysis of Section 4 and in the complexity analysis of the present section, we can assume that all partial degrees d_1, d'_1, d_2, d'_2 are of the same order. To keep experimentation and estimates manageable, we will assume that they all belong to a range $[2^k, 2^{k+1})$ for some $k \geq 2$. In the case of our complexity analysis, we will further assume that all d_1, d'_1, d_2, d'_2 are equal (or close) to a value d in such a range.

We call *degree cut-off FFT vs TFT* a value $d \in [2^k, 2^{k+1})$ such that the work (or algebraic complexity) of bivariate multiplication based on 2-D TFT is less than the one of bivariate multiplication based on 2-D FFT. Our objective in the sequel of this section is to determine the smallest possible cut-off for a given value of k .

To this end, we have developed a MAPLE package (available upon request) that manipulates polynomials with rational number coefficients and with k and 2^k as variables. We denote by $\mathbb{Q}[k, 2^k]$ the set of these objects. It satisfies all the usual algebraic rules on such expressions plus other operations targeting complexity analysis. For instance, our package computes symbolic logarithms of appropriate elements of $\mathbb{Q}[k, 2^k]$ and performs asymptotic majorations (i.e. majorations that hold for k big enough).

The table below gives the work for the algorithm of Section 2.3 when 1-D FFTs are performed by TFT. Note that for all d in the range $[2^k, 2^{k+1})$ the work of the Cooley-Tukey bivariate multiplication is $48 \times 4^k(3k + 7)$.

Table 1. Work of TFT-based bivariate multiplication

d	Work
2^k	$3(2^{k+1} + 1)^2(7 + 3k)$
$2^k + 2^{k-1}$	$81 k 4^k + 270 4^k + 54 k 2^k + 180 2^k + 9k + 30$
$2^k + 2^{k-1} + 2^{k-2}$	$\frac{441}{4} k 4^k + \frac{735}{2} 4^k + 63 k 2^k + 210 2^k + 9k + 30$
$2^k + 2^{k-1} + 2^{k-2} + 2^{k-3}$	$\frac{2025}{16} k 4^k + \frac{3375}{2} 4^k + \frac{135}{2} k 2^k + 225 2^k + 9k + 30$

Table 2. Degree cut-off estimate

$(c_1, c_2, c_3, c_4, c_5, c_6, c_7)$	Range for which this is a cut-off
$(1, 1, 1, 0, 0, 0, 0)$	$3 \leq k \leq 5$
$(1, 1, 1, 0, 1, 0, 0)$	$5 \leq k \leq 7$
$(1, 1, 1, 0, 1, 1, 0)$	$6 \leq k \leq 9$
$(1, 1, 1, 0, 1, 1, 1)$	$7 \leq k \leq 11$
$(1, 1, 1, 1, 0, 0, 0)$	$11 \leq k \leq 13$
$(1, 1, 1, 1, 0, 1, 0)$	$14 \leq k \leq 18$
$(1, 1, 1, 1, 1, 0, 0)$	$19 \leq k \leq 28$

Determining degree cut-off's (for FFT vs TFT) implies solving inequalities of the form $p > 0$ for $p \in \mathbb{Q}[k, 2^k]$. We achieve this by using standard techniques of real function analysis and we omit the details here.

For different values d of the form $2^k + c_1 2^{k-1} + \dots + c_7 2^{k-7}$ where each c_1, \dots, c_7 is either 0 or 1, we have compared the work of our bivariate multiplication based on either FFT or TFT. The above table lists some of our findings. These results suggest that for every range $[2^k, 2^{k+1})$ that occur in practice (see Section 4) a sharp (or minimal) degree cut-off is around $2^k + 2^{k-1} + 2^{k-2} + 2^{k-3}$. Our experimental results lead in fact to $2^k + 2^{k-1} + 2^{k-2}$ on 1 core, which seems to us coherent. Indeed our complexity analysis does not take several important factors such as memory management overhead and etc.

4 Efficient Implementation for Balanced Bivariate Multiplication

In this section we present our implementation techniques for FFT-based polynomial multiplication on multi-cores. These techniques avoid unnecessary calculations and reduce memory movement. Although we focus on balanced bivariate multiplication, our explanation covers to the general case, which we had to consider anyway. Indeed, optimizing our code in the general multivariate case was necessary to make a fair performance evaluation of our balanced bivariate multiplication and our reduction to it. We evaluate the performance of our implementation using VTune [4] and Cilkscreen [3]. We further compare the performance and determine experimentally the cut-off between TFT- and FFT-based bivariate multiplication for a large range of polynomials.

4.1 Implementation Techniques

As in Section 2 let $f \in \mathbb{K}[x_1, \dots, x_n]$ be a multivariate polynomial with degree d_i in x_i , for all $1 \leq i \leq n$. We represent f in a *dense recursive manner* w.r.t. the variable order $x_1 < \dots < x_n$. This encoding, which is similar to an n -dimensional matrix in row-major layout, is defined as follows:

- The coefficients of f are stored in a contiguous one-dimensional array B .
- The coefficient of the term $x_1^{e_1} \dots x_n^{e_n}$ is indexed by $\ell_1 \dots \ell_{n-1} e_n + \ell_1 \dots \ell_{n-2} e_{n-1} + \dots + \ell_1 e_2 + e_1$ in B , where $\ell_i = d_i + 1$ for all $1 \leq i \leq n$.

The parallelization of the multiplication algorithm in Section 2.3 takes advantage of the ease-of-use of the parallel constructs in Cilk++. For instance, when evaluating a polynomial by means of a n -dimensional FFT, the 1-D FFTs computed along the i -th dimension (for each $1 \leq i \leq n$) are parallelized by a `cilk_for` loop. Similarly, Step 2 of the multiplication algorithm is performed by a `cilk_for`. A special care is needed, however, for handling the large data sets and frequent memory access that this algorithm may involve. We described below the challenges and our solutions.

Data transposition. A number of $n - 1$ data transpositions is needed when performing the row-column n -dimensional algorithm. Data transposition is purely memory-bound and can be a bottleneck. For the 2-dimensional case, we use the cache-efficient code provided by Matteo Frigo. It employs a divide-conquer approach presented in [8] which fits the base case into the cache of the targeted machine. For the multi-dimensional case, we divide the problem into multiple 2-dimensional transpositions where Matteo Frigo's cache-efficient code can be applied. Consider for instance a trivariate polynomial representation, with dimension sizes of s_1, s_2, s_3 and where the variable ordering has to be changed from $x_1 < x_2 < x_3$ to $x_3 < x_2 < x_1$. First we exchange x_1 and x_2 by means of s_3 number of 2-dimensional transpositions of size $s_1 s_2$. This gives the order of $x_2 < x_1 < x_3$. To exchange x_2 and x_3 , we view variables x_2 and x_1 as one variable and group their coefficients vector into one of size $\ell = s_2 s_1$. Then, one 2-dimensional transposition of size ℓs_3 will suffice. The resulting variable order is now $x_3 < x_2 < x_1$.

Avoiding unnecessary calculations and reducing memory movement. We first discuss the implementation of the algorithm of Section 2.3 when the 1-D FFTs are performed by the (radix 2) Cooley-Tukey algorithm. We allocate two workspaces A and B each with size $s = s_1 \cdots s_n$ for the evaluation of f and g respectively. To prepare for the interpolation of the product $f g$, the coefficient data of f and g are scattered to the appropriate positions of A and B . One can make these data movements first and then evaluate the polynomials. We use a more cache-efficient way instead. On the evaluation of the first variable of f , we copy in parallel each of the $(d_2 + 1) \cdots (d_n + 1)$ number of coefficient vectors of size $d_1 + 1$ from f to the corresponding vector of size s_1 in A and continue the evaluation of this vector in A by a 1-D FFT in place. This method improves the data-locality. It also saves from not doing FFTs on $(s_2 \cdots s_n) - (d_2 + 1) \cdots (d_n + 1)$ number of size s_1 vectors of zeros. We proceed similarly with g .

For the TFT version of the algorithm of Section 2.3 we have realized two implementations. One implementation is similar to the one above and we call it “in-place”. Each data transposition has to be done for the size s , but the number of 1-D FFTs in an evaluation or interpolation are bounded by $ps = (d_1 + d'_1 + 1) \cdots (d_n + d'_n + 1)$, the size of the product. Our second implementation is “out-of-place”. We only allocate one extra workspace C of size ps . C is used for the evaluation of g . We will use the space of the product for the evaluation of f , and the result will be in the right place. Savings can be gained on the evaluation of the first variable in the same way as above. The difference is as follows. To evaluate a vector v_i of size $d_i + d'_i + 1$ in variable x_i , we allocate a temporary vector t_i of size s_i as a workspace and copy the data in v_i to s_i and then perform 1-D TFT in t_i ; t_i is freed after use.

Our benchmark shows that the “out-of-place” TFT-based method is more efficient for balanced problems (that is when the partial degrees of the product are equal); moreover the performance of this approach becomes even better when the number of cores increases. However, for problems with unequal partial degrees in the product, the “in-place” TFT-based method works better. We will study in a future work the causes of this behavior.

4.2 Performance Evaluation

We use VTune [4] and Cilkscreen [3] to evaluate the performance of our implementation. We measure the instruction, cache and parallel efficiency on 8 processors for the multiplication of bivariate polynomials with partial degrees in the range of [2047, 4096].

Table 3 lists a selection of events and ratios reported by VTune for FFT-based and TFT-based methods respectively. Due to the availability of VTune in our laboratory, these measurements are done on a 8-core machine with 8 GB memory. Each processor is Intel Xeon X5460 @3.16GHz and has 6144 KB of L2 cache.

For all the tested problems by either FFT or TFT method, their clocks per instructions retired (CPI) is around 0.8. Their L2 cache miss rates are below 0.0008. The very small modified data sharing ratios (less than 0.00025) imply that, chances of threads racing on using and modifying data laid in one cache line are very low. However, TFT-based method use about three times less number of instructions retired than FFT-based for problems which are worst cases for FFT, such as (2048, 2048), (2048, 4096) and (4096, 4096). These account for the better timing of TFT-based method for a certain range of degrees, shown in the figures of next section.

We use Cilkscreen to estimate the parallelism of these running instances by measuring their *work* and *span*; recall that we rely on the multi-threaded parallelism model introduced in [2]. The data measured by Cilkscreen are summarized in Table 4. The

Table 3. Performance evaluation by VTune for TFT-based and FFT-based method

Method	d_1	d_2	INST- RETIRED ($\times 10^9$)	Clocks per L2 cache instructions retired (CPI)	Modified data sharing ratio ($\times 10^{-3}$)	Time on 8 cores (s)
TFT-based	2047	2047	44	0.794	0.423	0.215
	2048	2048	52	0.752	0.364	0.163
	2047	4095	89	0.871	0.687	0.181
	2048	4096	106	0.822	0.574	0.136
	4095	4095	179	0.781	0.359	0.141
	4096	4096	217	0.752	0.309	0.115
FFT-based	2047	2047	38	0.751	0.448	0.106
	2048	2048	145	0.652	0.378	0.073
	2047	4095	79	0.849	0.745	0.122
	2048	4096	305	0.765	0.698	0.094
	4095	4095	160	0.751	0.418	0.074
	4096	4096	622	0.665	0.353	0.060

Table 4. Performance evaluation by Cilkscreen for TFT-based and FFT-based method

Method	d_1	d_2	Work	Span/ Burdened ($\times 10^9$)	Parallelism/ Burdened parallelism	Speedup estimates		
	4P	8P	16P					
TFT-based	2047	2047	45	0.613/0.614	74.18/74.02	3.69-4	6.77-8	11.63-16
	2048	2048	53	0.615/0.616	86.35/86.17	3.74-4	6.96-8	12.22-16
	2047	4095	109	0.118/0.118	92.69/92.58	3.79-4	7.09-8	12.54-16
	2048	4096	125	1.184/1.185	105.41/105.27	3.80-4	7.19-8	12.88-16
	4095	4095	193	2.431/2.433	79.29/79.24	3.71-4	6.86-8	11.89-16
	4096	4096	223	2.436/2.437	91.68/91.63	3.76-4	7.03-8	12.43-16
FFT-based	2047	2047	40	0.612/0.613	65.05/64.92	3.64-4	6.59-8	11.08-16
	2048	2048	155	0.619/0.620	250.91/250.39	3.80-4	7.50-8	14.55-16
	2047	4095	98	1.179/1.180	82.82/82.72	3.77-4	6.99-8	12.23-16
	2048	4096	383	1.190/1.191	321.75/321.34	3.80-4	7.60-8	14.82-16
	4095	4095	169	2.429/2.431	69.39/69.35	3.66-4	6.68-8	11.35-16
	4096	4096	392	2.355/2.356	166.30/166.19	3.80-4	7.47-8	13.87-16

Table 5. Sizes and cut-offs of three sets of problems

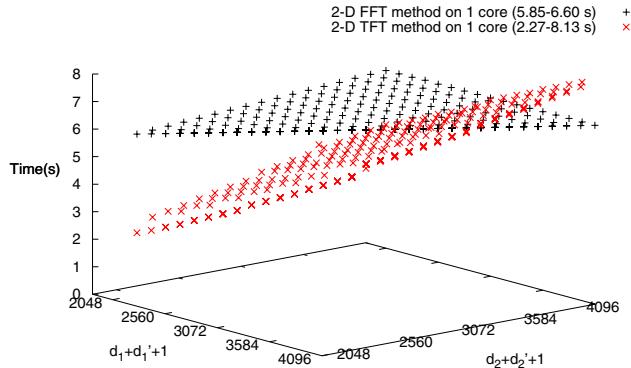
No.	Input degree range	Product size range	Size cut-off on			
			1 core	8 cores	12 cores	16 cores
1	256-511	263169-1046529	786596	814012	861569	
2	1023-2047	4198401-16769025	12545728	13127496	14499265	16433645
3	4095-8191	67125249-268402689	202660762	207958209	227873850	257370624

parallel overhead appears very low and the burdened parallelism is nearly equal to the expected parallelism. The speedup factors that we obtain, reported in the next section, are as good as the estimated ones.

4.3 Cut-Off between TFT- and FFT-Based Methods

We compare the performances of the FFT- and TFT-based bivariate multiplication. More precisely and as discussed in Section 3, we consider that all partial degrees d_1, d_2, d'_1, d'_2 are between two consecutive powers of 2 and we would like to determine for which degree patterns the TFT approach outperforms the FFT one. We study the following three degree ranges: [256, 512], [1024, 2048] and [4096, 8192]. In general, we use the “in-place” implementations of FFT- and TFT-based methods. When the partial degrees of the product are equal, the “out-of-place” TFT-based method is used.

The sizes and the cut-offs on 1, 8, 12 and 16 cores for the three sets of problems are summarized in Table 5. Table 6 lists the timings, the speedup factors, the percentages of the size range and the ratio by which TFT or FFT is superior for the three sets of problems on 1, 8, 12 and 16 cores. To provide an insight view of the benchmarks, we display the complete timing results and their cut-off regression w.r.t. the size of the product for the range of [1024, 2048] on 1, 8 and 16 cores in Figures 11 to 16.

**Fig. 1.** Timing of bivariate multiplication for input degree range of $[1024, 2048)$ on 1 core**Table 6.** Cut-off details between TFT-based and FFT-based bivariate multiplication

No.	#cores	TFT-based Method			FFT-based Method		
		Time (s)	Speedup factor	Faster portion	Time (s)	Speedup factor	Faster portion
1	1	0.120-0.419		75%	2.55-1.0	0.305-0.338	
	8	0.020-0.065	5.5-6.7	78%	2.50-1.0	0.050-0.055	6.0-6.3
	12	0.015-0.048	5.5-9.0	82%	2.73-1.0	0.038-0.044	7.4-8.2
2	1	2.27-8.13	2.58-1.0	75%		5.85-6.60	
	8	0.309-1.08	6.8-7.6	78%	2.61-1.0	0.806-0.902	7.2-7.3
	12	0.224-0.779	8.2-11.0	86%	2.77-1.0	0.613-0.707	9.3-9.8
	16	0.183-0.668	7.8-14.1	98%	3.18-1.0	0.588-0.661	9.6-10.8
3	1	42.2-154.3		76%	2.63-1.0	110.9-123.2	
	8	5.52-20.07	6.8-7.8	77%	2.69-1.0	14.82-16.57	7.4-7.6
	12	3.75-14.10	9.0-11.4	85%	2.92-1.0	10.96-12.72	9.9-10.3
	16	3.09-11.36	9.5-14.9	96%	3.12-1.0	9.55-11.02	11.0-12.0

Figures 1 to 6 reveal clearly the different performances of FFT- and TFT-based methods for the problems in set 2 with partial degrees in the range of $[1024, 2048)$, which is $[2^{10}, 2^{11})$. Here, the timing of FFT-based method is about the same for all the problems, but the timing of TFT-based method is correlated to the size of the partial degrees. This result agrees with our complexity analysis reported in Expressions (1) and (2) in Section 2.3. Expression (1) indicates that the work for multiplying any pair of n -variate polynomials with partial degrees d_i and d'_i in a range of $[2^k, 2^{k+1})$ by FFT-based method is constant, and determined by the value of s , which is $2^{n(k+2)}$. This reflects the well-known *staircase* phenomenon of FFT. Meanwhile, the work of TFT-based method grows linearly with $\prod_{i=1}^n (d_i + d'_i + 1)$, as indicated by Expression (2). Therefore, in a certain lower range of $[2^k, 2^{k+1})$, TFT-based method performs better.

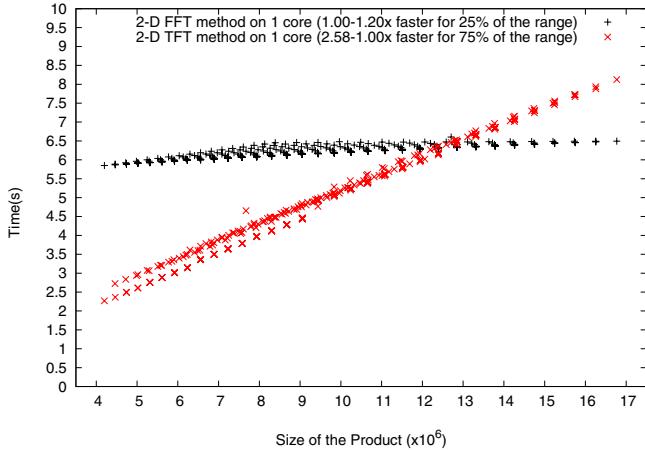


Fig. 2. Size cut-off for input degree range of [1024, 2048) on 1 core

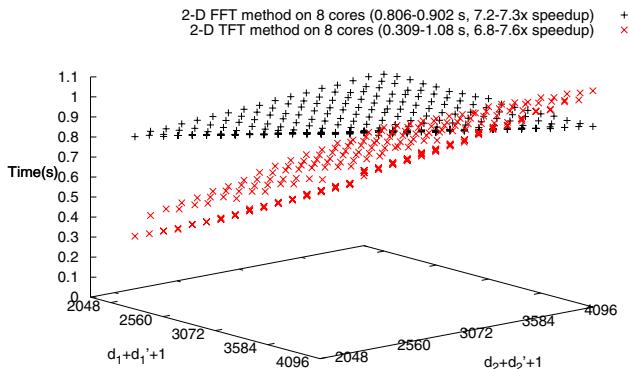
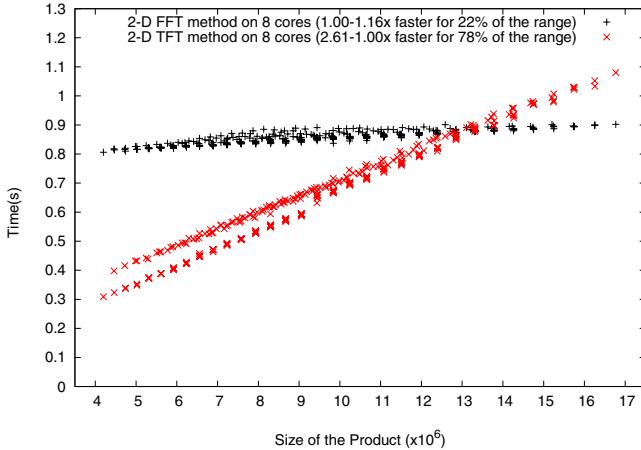
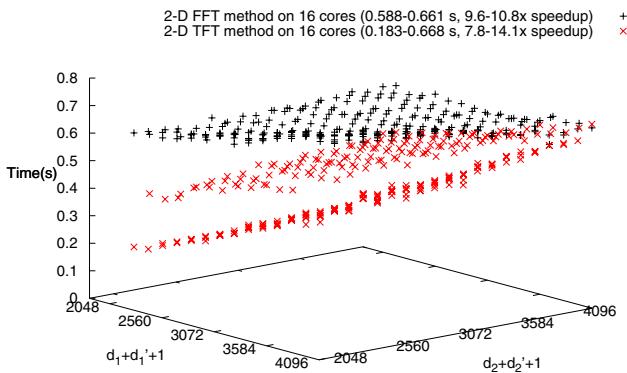


Fig. 3. Timing of bivariate multiplication for input degree range of [1024, 2048) on 8 cores

Overall, both FFT- and TFT-based methods show good speedup factors on 8 to 16 cores, with peak performance for the latter. The cut-off criteria are similar for all degree ranges, independent of the magnitude of the degrees. In a degree range of $[2^k, 2^{k+1})$, the percentage of problems for which the TFT-based method outperforms FFT's increases with the number of cores. On one core, the TFT-based method is better for the first 75% of the sizes by at most a factor of 2.6. This is coherent to the theoretical analysis result in Section 3, taking into account the memory management overhead in our implementation. On 16 cores, the TFT-based method is superior for up to 98% of the problem range by a maximum factor of 3.2. The mechanism that favors the TFT-based method on multi-cores will be studied further.

**Fig. 4.** Size cut-off for input degree range of [1024, 2048) on 8 cores**Fig. 5.** Timing of bivariate multiplication for input degree range of [1024, 2048) on 16 cores

5 Parallelism Estimates for Normal Form Computations

The recursive structure of the procedure NormalForm_n in Section 2.5 offers opportunities for concurrent execution. Moreover, this procedure relies on multivariate multiplication and we can hope to increase parallelism by relying on our parallel multiplication. Estimating this extra parallel speedup factor is a fundamental problem, as discussed in the introduction. This section offers a first answer based on complexity analysis meanwhile Section 6 will provide experimental results.

We first consider the span (or parallel running time) of the algorithms in Section 2 by means of the multi-threaded programming model of [9]. This model, however, does not explicitly cover parallel for-loops, which are needed for both multiplication and normal form computations.

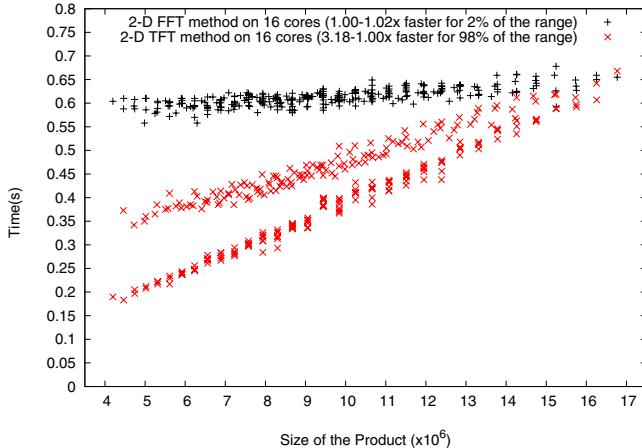


Fig. 6. Size cut-off for input degree range of [1024, 2048] on 16 cores

Following the way a `cilk_for` loop is implemented in the `cilk++` language [3], we assume that the span of a for-loop of the form

```
for i from 1 to n do BODY(i); end for;
```

is bounded by $O(\lg(n)S)$ where S is the maximum span of `BODY(i)` for i in the range $1..n$. Consequently the span of a nested for-loop

```
for j in 1..m do
    for i from 1 to n do BODY(i); end for;
end for;
```

is bounded by $O(\lg(n)\lg(m)S)$.

Defining $s = s_1 \cdots s_n$ and $\ell = \prod_{i=1}^n \lg(s_i)$, it is easy to check that the span of the multiplication algorithm of Section 2.3 is

$$3 \sum_{i=1}^n \left(\prod_{j \neq i} \lg(s_j) \right) s_i \lg(s_i) + 3 \prod_{i=1}^n \lg(s_i) = 3\ell \left(\sum_{i=1}^n s_i + 1 \right) \quad (3)$$

operations in \mathbb{K} , when the Cooley-Tukey algorithm is used for 1-D FFTs. This estimate becomes

$$3 \sum_{i=1}^n \frac{\ell}{\lg(s_i)} (d_i + d'_i + 1) (\lg(s_i) + 1) + 3 \prod_{i=1}^n \lg(d_i + d'_i + 1), \quad (4)$$

which is the same order of magnitude as

$$3\ell \sum_{i=1}^n (d_i + d'_i + 1) + 3 \prod_{i=1}^n \lg(d_i + d'_i + 1), \quad (5)$$

when all the s_i become large.

We turn now to the span estimates for the procedure NormalForm_i when applied to f and $\{g_1, \dots, g_i\}$ where g_1, \dots, g_i is a reduced monic triangular set of $\mathbb{K}[x_1, \dots, x_i]$ (see Section 2) and f is a polynomial of $\mathbb{K}[x_1, \dots, x_i]$. In practice, the partial degree of f w.r.t x_i is at most $2\delta_i - 2$, for all $1 \leq i \leq n$. Indeed, the polynomial f is often the product of two polynomials a and b which are reduced w.r.t. the reduced monic triangular set $\{g_1, \dots, g_i\}$, that is, which satisfy $\deg(a, x_j) < \delta_j$ and $\deg(b, x_j) < \delta_j$ for all $1 \leq j \leq i$.

Define $\underline{\delta}_i = (\delta_1, \dots, \delta_i)$. Let us denote by $W_M(\underline{\delta}_i)$ and $S_M(\underline{\delta}_i)$ the work and span of a multiplication algorithm applied to h and g_i where h satisfies $\deg(h, x_j) < \delta_j$ for $1 \leq j < i$ and $\deg(h, x_i) \leq 2\delta_i - 2$. Let also $S_{NF}(\underline{\delta}_i)$ be the span of NormalForm_i applied to f and $\{g_1, \dots, g_i\}$. If the procedure NormalForm_i is run with a serial multiplication, then we have:

$$S_{NF}(\underline{\delta}_i) = 3\ell_i S_{NF}(\underline{\delta}_{i-1}) + 2W_M(\underline{\delta}_i) + \ell_i \quad (6)$$

where $\ell_i = \prod_{j=1}^i \lg(\delta_j)$. Similarly, if the procedure NormalForm_i is run with a parallel multiplication, we obtain:

$$S_{NF}(\underline{\delta}_i) = 3\ell_i S_{NF}(\underline{\delta}_{i-1}) + 2S_M(\underline{\delta}_i) + \ell_i. \quad (7)$$

Neglecting logarithmic factors and denoting by d the maximum of δ_i for all $1 \leq i \leq n$, the span $S_{NF}(\underline{\delta}_i) \in O(3^n d^n)$ if a serial multiplication is used, otherwise $S_{NF}(\underline{\delta}_i) \in O(3^n d)$ if a parallel multiplication is used. Since the work $W_{NF}(\underline{\delta}_i) \in O(4^n d^n)$ (again neglecting logarithmic factors) this implies that work, span and parallelism (i.e. the ratio of work divided by span) are all exponential in the number of variables. This suggests that obtaining efficient parallel implementation of the procedure NormalForm_i is interesting but also challenging.

In Tables 7 and 8, the span of NormalForm_i is computed for $i = 1, 2, 3$ and $\delta_1 = \dots = \delta_i = d$ with $d \in \{2^k, 2^k + 2^{k-1}\}$. For $i = 1$, the spans of NormalForm_i with or without parallel multiplication are essentially the same. Indeed we assume that 1-D

Table 7. Span of TFT-based normal form for $\underline{\delta}_i = (2^k, \dots, 2^k)$

i	With serial multiplication	With parallel multiplication
1	$18k2^k + 442^k + 10k + 22$	$12k2^k + 242^k + 11k + 20$
2	$72k4^k + 1684^k + o(4^k)$	$60k^22^k + 168k^22^k + 962^k + o(2^k)$
3	$216k8^k + 4968^k + o(8^k)$	$216k^32^k + 720k^22^k + 720k2^k + 2882^k + o(2^k)$

Table 8. Span of TFT-based normal form for $\underline{\delta}_i = (2^k + 2^{k-1}, \dots, 2^k + 2^{k-1})$

i	With serial multiplication	With parallel multiplication
1	$27k2^k + 932^k + 10k + 32$	$18k2^k + 542^k + 11k + 27$
2	$162k4^k + 5404^k + o(4^k)$	$90k^22^k + 396k^22^k + 3782^k + o(2^k)$
3	$729k8^k + 24038^k + o(8^k)$	$324k^32^k + 1836k^22^k + 3186k2^k + 17822^k + o(2^k)$

FFTs are run serially. The slight gain is explained by the fact in **Step 1** of the algorithm of Section 2.5 One can evaluate the two input polynomials concurrently.

For $i = 2$, the gain obtained from the use of a parallel multiplication is asymptotically in the order of $\Theta(2^k/k)$. For $i = 3$, this becomes $\Theta(4^k/k^2)$. This suggests that a parallel multiplication code (even under the 1-D FFT black box assumption) can speedup substantially a parallel code for NormalForm_i with $i \geq 2$.

In Table 9 we provide the span of NormalForm_i for another degree pattern, namely for $\delta_i = 2^k$ and $\delta_{i-1} = \dots = \delta_1 = 1$. This configuration is actually the general one for the polynomials describing the symbolic solutions of polynomial systems with finitely many solutions. We call it *Shape Lemma* after the landmark paper [1] where this degree pattern was formally studied and from which the terminology is derived.

In Table 10 we provide the limit of the parallelism of NormalForm_i when k goes to $+\infty$ (that is the ratio between work and span) for the same degree patterns as in Table 9.

Table 9. Span of TFT-based normal form for $\underline{\delta}_i = (2^k, 1, \dots, 1)$

i	With serial multiplication	With parallel multiplication
2	$144 k 2^k + 642 2^k + 76 k + 321$	$72 k 2^k + 144 2^k + 160 k + 312$
4	$4896 k 2^k + 45028 2^k + 2488 k + 22514$	$1296 k 2^k + 2592 2^k + 6304 k + 12528$
8	$3456576 k 2^k + 71229768 2^k + o(2^k)$	$209952 k 2^k + 419904 2^k + o(2^k)$

Table 10. Parallelism estimates of TFT-based normal form for $\underline{\delta}_i = (2^k, 1, \dots, 1)$

i	With serial multiplication	With parallel multiplication
2	$13/8 \simeq 2$	$13/4 \simeq 3$
4	$1157/272 \simeq 4$	$1157/72 \simeq 16$
8	$5462197/192032 \simeq 29$	$5462197/11664 \simeq 469$

Table 9 suggests that for *Shape Lemma* degree patterns and for a fixed number of variables, the extra speedup factor provided by a parallel multiplication (w.r.t. a serial one) is upper bounded by a constant. This does not imply that using parallel multiplication in a parallel normal form for *Shape Lemma* degree patterns would be of limited practical interest. Table 10 suggests that a parallel multiplication can indeed increase the parallelism of NormalForm_i , in particular when the number of variables is large.

These complexity estimates do not take into account parallel overhead. In the case of NormalForm_i , those are potentially large. Indeed, after Steps 1, 4 and 6 of NormalForm_i , there is a synchronization point for $(\delta_i + 1)$ threads, namely the $(\delta_i + 1)$ recursive calls to NormalForm_{i-1} . Observe also that the number of synchronization points of NormalForm_i is 3^{i-1} . This puts a lot of “burden” on the parallelism of this procedure. Section 6 will tell how much can really be achieved in practice.

6 Efficient Parallel Computation of Normal Forms

We present now our experimental results for efficient parallelization of normal forms. Our key techniques include reducing multivariate multiplication to bivariate and composing the parallelism of bivariate multiplications with that at the level of the normal form algorithm, as discussed in Section 5. We study three typical degree patterns, illustrated in Figures 7, 8 and 9. All the benchmarks show the important role of parallel bivariate multiplication in improving the performance of normal form computations.

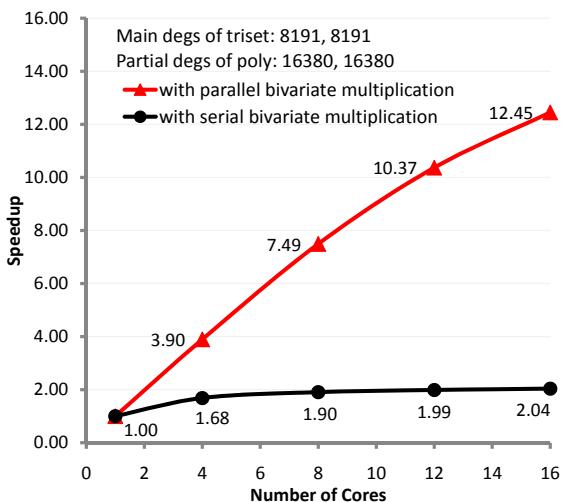


Fig. 7. Normal form computation of a large bivariate problem

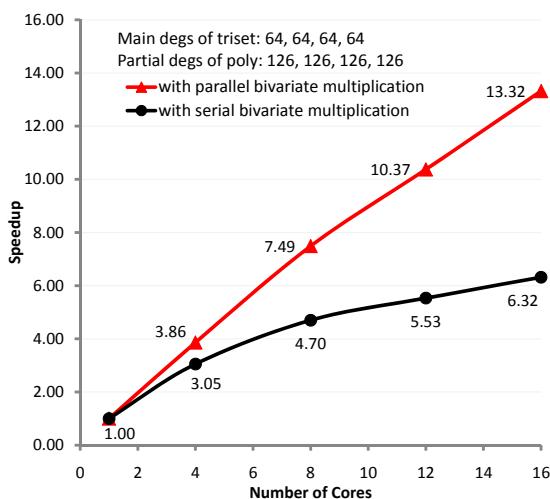


Fig. 8. Normal form computation of a medium-sized 4-variate problem

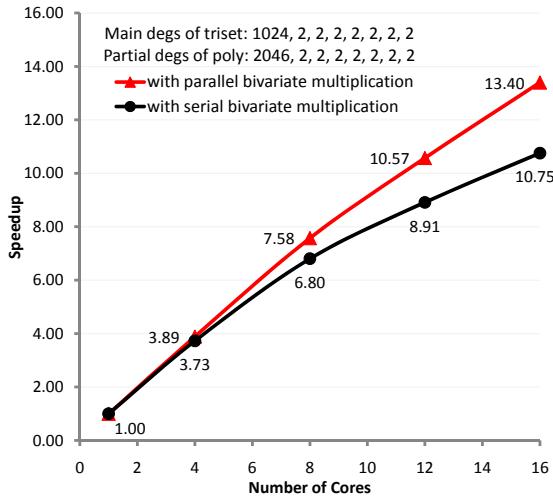


Fig. 9. Normal form computation of an irregular 8-variate problem

Figure 7 displays the speedup factors for computing the normal form of a bivariate problem with serial bivariate multiplication and with parallel bivariate multiplication. The main degrees of the polynomials of a triangular set are 8191 and 8191, and the partial degrees of the polynomial under simplification are 16380 and 16380. The speedup of normal form computation without parallel multiplication is very poor, about 2.0 on 16 cores. The parallelization of the bivariate multiplications helps improving the performance significantly, by a factor of 6.

Figure 8 demonstrates the 2.0 times of improvement contributed by the parallel bivariate multiplications involved in the normal form computation of a medium-sized 4-variate problem, where all the main degrees of the triangular set are equal to 64, and all the partial degrees of the polynomial to be reduced are 126. The 8-variate problem, described in Figure 9 with main degree pattern of “1024, 2, 2, 2, 2, 2, 2, 2”, shows a speedup of 10.75 on 16 cores without parallel bivariate multiplication. Composed with parallel bivariate multiplication it can achieve a more satisfactory speedup of 13.4.

These results show that, when the number of variables is small, say 2, the parallelism of our normal form routine can be small. However, if the input polynomial degrees are large enough, parallel multiplication can increase the overall parallelism substantially. In the Shape Lemma case, when the number of variables is large, say 8, our normal form routine already possesses a high parallelism. Hence, even though parallel multiplication cannot help as much as in the previous case, the combination of the two parallel code brings again high performance.

7 Concluding Remarks

We have reported implementation strategies for FFT-based dense polynomial arithmetic targeting multi-cores. We have extended our preliminary study [19] dedicated to

multiplication leading to a complete set of efficient routines for polynomial arithmetic operations, including normal form computations.

Since balanced bivariate multiplication is the kernel to which most of these routines reduce, we have conducted an in-depth study on the implementation techniques for this operation. Our performance analysis by VTune and Cilkscreen show that our implementations have good instruction and cache efficiency, and good parallelism as well. In particular we have determined cut-off criteria between two variants of this balanced bivariate multiplication based respectively on Cooley-Tukey FFT and the Truncated Fourier Transform on multi-cores. The cut-off criteria are similar for all degree ranges that we have tested. However, for a fixed degree range of the form $[2^k, 2^{k+1})$, the percentage of problems for which TFT-based method outperforms FFT's increases with the number of cores.

We have explained why the parallelization of normal form computation is challenging and also of great importance in symbolic computation. We have shown that, not only efficient parallel multiplication can improve the performance of parallel normal form computation, but also that this composition is necessary for parallel normal form computation to reach peak performance on all input patterns that we have tested.

For both problems of optimizing balanced bivariate multiplication and performing efficient parallel computation of normal forms, we have combined theoretical and empirical analyses. The former could not provide a precise answer due to simplification hypotheses but helped narrowing the pattern ranges for the latter analysis.

Nevertheless, we would like to obtain more “realistic” results through complexity analysis. In the context of this study, this means being able to better take parallel overhead into account: A subject of future research.

Another future work is the development of higher-level algorithms on top of the basic polynomial algebra subroutines (BPAS) presented in this paper. Our driving application is the solving of polynomial systems symbolically. Our next step toward this goal is the parallelization of polynomial GCDs modulo regular chains, following the work of [15].

Acknowledgements. This work was supported by NSERC and the MITACS NCE of Canada, and NSF under Grants 0540248, 0615215, 0541209, and 0621511. We are also very grateful for the help of Professor Charles E. Leiserson, Dr. Matteo Frigo and all other members of SuperTech Group at CSAIL MIT and Cilk Arts.

References

1. Becker, E., Mora, T., Marinari, M.G., Traverso, C.: The shape of the shape lemma. In: Proc. of ISSAC 1994, pp. 129–133. ACM Press, New York (1994)
2. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. In: IEEE FOCS 1994 (1994)
3. Cilk Arts. Cilk++, <http://www.cilk.com/>
4. Intel Company. Intel VTune Performance Analyzer 9.1 for Linux, <http://www.intel.com/>
5. Cook, S.: On the minimum computation time of function. PhD thesis, Harvard Univ. (1966)
6. Cooley, J., Tukey, J.: An algorithm for the machine calculation of complex Fourier series. Math. Comp. 19, 297–301 (1965)

7. Cox, D., Little, J., O’Shea, D.: Using Algebraic Geometry. Graduate Text in Mathematics, vol. 185. Springer, New York (1998)
8. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: 40th Annual Symposium on Foundations of Computer Science, pp. 285–297 (1999)
9. Frigo, M., Leiserson, C.E., Randall, K.H.: The implementation of the cilk-5 multithreaded language. In: ACM SIGPLAN (1998)
10. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge Univ. Press, Cambridge (1999)
11. van der Hoeven, J.: Truncated Fourier transform. In: Proc. ISSAC 2004. ACM Press, New York (2004)
12. Huss-Lederman, S., Jacobson, E.M., Tsao, A., Turnbull, T., Johnson, J.R.: Implementation of Strassen’s algorithm for matrix multiplication. In: Supercomputing 1996: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM), Washington, DC, USA, p. 32. IEEE Computer Society, Los Alamitos (1996)
13. Kung, H.T.: On computing reciprocals of power series. Numerische Mathematik 22, 341–348 (1974)
14. Li, X., Moreno Maza, M.: Multithreaded parallel implementation of arithmetic operations modulo a triangular set. In: Proc. PASCO 2007, pp. 53–59. ACM Press, New York (2006)
15. Li, X., Moreno Maza, M., Pan, W.: Computations modulo regular chains. In: Proc. of ISSAC 2009, pp. 239–246. ACM Press, New York (2009)
16. Li, X., Moreno Maza, M., Rasheed, R., Schost, É.: The modpn library: Bringing fast polynomial arithmetic into maple. In: MICA 2008 (2008)
17. Li, X., Moreno Maza, M., Schost, É.: Fast arithmetic for triangular sets: From theory to practice. In: Proc. ISSAC 2007, pp. 269–276. ACM Press, New York (2007)
18. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)
19. Moreno Maza, M., Xie, Y.: Balanced dense polynomial multiplication on multicore. In: Proc. PDCAT 2009, Hiroshima, Japan (2009)
20. Sieveking, M.: An algorithm for division of powerseries. Computing 10, 153–156 (1972)

Case Study of Scientific Data Processing on a Cloud Using Hadoop

Chen Zhang¹, Hans De Sterck², Ashraf Aboulnaga¹,
Haig Djambazian³, and Rob Sladek³

¹ David R. Cheriton School of Computer Science, University of Waterloo,
Ontario, N2L 3G1, Canada

² Department of Applied Mathematics, University of Waterloo,
Ontario, N2L 3G1, Canada

³ McGill University and Genome Quebec Innovation Centre, Montreal,
Quebec, H3A 1A4, Canada

Abstract. With the increasing popularity of cloud computing, Hadoop has become a widely used open source cloud computing framework for large scale data processing. However, few efforts have been made to demonstrate the applicability of Hadoop to various real-world application scenarios in fields other than server side computations such as web indexing, etc. In this paper, we use the Hadoop cloud computing framework to develop a user application that allows processing of scientific data on clouds. A simple extension to Hadoop’s MapReduce is described which allows it to handle scientific data processing problems with arbitrary input formats and explicit control over how the input is split. This approach is used to develop a Hadoop-based cloud computing application that processes sequences of microscope images of live cells, and we test its performance. It is discussed how the approach can be generalized to more complicated scientific data processing problems.

1 Introduction

The concept of ‘cloud computing’ is currently receiving considerable attention, both in the research and commercial arenas. While cloud computing concepts are closely related to the general ideas and goals of grid computing, there are some specific characteristics that make cloud computing promising as a paradigm for transparently scalable distributed computing. In particular, two important properties that many cloud systems share are the following:

1. Clouds provide a homogeneous operating environment (for instance, identical operating system (OS) and libraries on all cloud nodes, possibly via virtualization).
2. Clouds provide full control over dedicated resources (in many cases, the cloud is set up such that the application has full control over exactly the right amount of dedicated resources, and more dedicated resources may be added as the needs of the application grow).

While these two properties lead to systems that are less general than what is normally considered in the grid computing context, they significantly simplify the technical implementation of cloud computing solutions, possibly to the level where feasible, easily deployable technical solutions can be worked out. The fact that cloud computing solutions, after only a short time, have already become commercially viable would point in that direction. Indeed, the first property above removes the complexity of dealing with versions of application code that can be executed in a large variety of software operating environments, and the second property removes the complexity of dealing with resource discovery, queuing systems, reservations, etc., which are the characteristics of shared environments with heterogeneous resources.

Cloud computing is thus a promising paradigm for transparently scalable distributed computing. It was pioneered in large-scale web application processing: a well-known and highly successful example is Google's web processing and hosting system. Cloud computing is now also being used and explored extensively for enterprise applications. This paper is concerned with the application of cloud computing to large-scale scientific data processing, an area which is relatively unexplored so far.

It is useful to distinguish three software layers in clouds:

1. The OS (possibly virtualized).
2. A cloud computing framework, often including an execution environment, a storage system, and a database-like capacity.
3. A user application built on top of layers 1 and 2.

Google's system is a well-known cloud computing framework, and several commercial cloud computing frameworks have recently appeared on the market, including products from GigaSpaces [19], Elastra [16], etc. A well-known commercial cloud provider is Amazon [3].

In this paper, we focus on cloud computing environments similar to Google's system, which was designed for scalable and reliable processing and hosting of the very large data sets that are provided by the world's largest 'information organization' company. Three major components in Google's web processing system are:

1. MapReduce, a scalable and reliable programming model and execution environment for processing and generating large data sets.
2. Google File System (GFS), a scalable and reliable distributed file system for large data sets.
3. BigTable, a scalable and reliable distributed storage system for sparse structured data.

Google's system is tailored to specific applications. GFS can deal efficiently with large input files that are normally written once and read many times. MapReduce handles large processing jobs that can be parallelized easily: the input normally consists of a very long sequence of atomic input records that can be processed independently, at least in the first phase. Results can then

be collected (reduced) in a second processing phase, with file-based key-value pair communication between the two phases. MapReduce features a simple but expressive programming paradigm, which hides parallelism and fault-tolerance. The large input files can be split automatically into smaller files that are processed on different cloud nodes, normally by splitting files at the boundaries of blocks that are stored on different cloud nodes. These split files are normally distributed over the cloud nodes, and MapReduce attempts to move computation to the nodes where the data records reside. Scalability is obtained by the ability to use more resources as demand increases, and reliability is obtained by fault-tolerance mechanisms based on replication and redundant execution.

In this paper, we use Hadoop [4], which is an open source implementation of a subset of the Google system described above. The three Hadoop components that are analogous to Google's components described above are:

1. Hadoop's MapReduce.
2. Hadoop's Distributed File System (DFS).
3. The HBase storage system for sparse structured data.

Hadoop is used by companies like Yahoo and Facebook, and is also widely used as an instruction tool for cloud computing education.

In this paper, we use the Hadoop cloud computing framework to develop a simple user application that allows processing of scientific data of a certain type on clouds. Our paper is mainly an application paper, exploring the use of Hadoop-based cloud computing for a scientific data processing problem. At the same time, we describe how we developed a small extension to Hadoop's MapReduce which allows it to handle scientific data processing applications, and we report on our experience with performance studies.

Our paper focuses on a real scientific data processing case study: we develop a cloud application for a specific project that requires large-scale biological image processing. Large amounts of data are generated that need to be organized systematically in the cloud, and various types of data processing have to be performed (most algorithms use MATLAB). The application, built on top of Hadoop, allows users to submit data processing jobs to the cloud. At expected full production levels, the scale of the problem calls for the type of scalable and reliable solution platform that cloud computing can provide.

We find that we can use Hadoop without much modification: the only required change is to MapReduce input formats and the splitting of the input. Hadoop's MapReduce input is normally file-based, and we extend this to more general types of inputs, like file folders or database (DB) tables. More importantly, Hadoop normally splits files in parts that have equal binary size, irrespective of the boundaries between atomic input records. We modify this such that the user can easily provide a method that splits the input along boundaries between atomic input records. For example, in our application the atomic input records are folders with image files, and we allow the user to specify the number of groups these input folders have to be split in. In another example the atomic input records could be the rows of a DB table, and the user would be allowed to specify how many groups of rows the table needs to be split in. Note that

these are small modifications of a practical nature, especially since Google's MapReduce allows for such user-defined splitting methods as well. Nevertheless, Hadoop does not currently allow for this type of splitting in an easy way, and we describe an easy solution to this problem.

The scientific data processing problem considered in this paper is simple: the workload is divisible, without the need for communication between tasks. In this simple example there is only one processing phase, and thus there is no need to use the 'reduce' phase of Hadoop's MapReduce. Nevertheless, our cloud solution relies on many of the other features offered by the cloud concept and Hadoop, including scalability, reliability, fault-tolerance, easy deployability, etc.

As discussed in more detail at the end of this paper, our approach can be generalized easily to more complicated scientific data processing jobs (such as jobs with input data stored in a relational database, jobs that require a reduce phase after the map phase, scientific workflows, etc.). In this paper we focus on a simple real-world test case to demonstrate the applicability of cloud computing with Hadoop to scientific data processing in principle, and to relate our experiences in attempting to do this, and we leave the more complicated scientific computing applications for future work.

The rest of this paper is organized as follows. In the next section, we describe the biomedical image processing problem for which we develop a cloud application in this paper. The design of our system and the modifications to Hadoop's MapReduce are described in Section III, followed by performance analysis and system refinements in Section IV. This is followed by a discussion section (Section V) and an overview of related work in Section VI. Section VII formulates conclusions and describes future work.

2 Case Study

In this paper, we develop a Hadoop-based cloud computing application that processes sequences of microscope images of live cells. This project is a collaboration between groups at Genome Quebec/McGill University in Montreal, and at the University of Waterloo. We first describe the general context of the project, followed by a detailed description of the image processing tasks that will be considered later on in the paper.

2.1 General Description of Biomedical Image Processing Problem

Our goal is to study the complex molecular interactions that regulate biological systems. To achieve this we are developing an imaging platform to acquire and analyze live cell data at single cell resolution from populations of cells studied under different experimental conditions. The key feature of our acquisition system is its capability to record data in high throughput, both in the number of images that can be captured for a single experimental condition and the number of different experimental conditions that can be studied simultaneously. This is achieved by using an automated bright field and epifluorescence microscope in

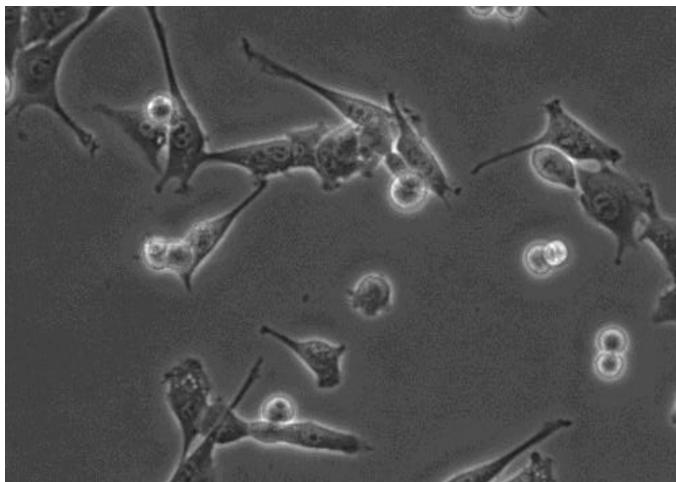


Fig. 1. Single microscope image with about two dozen cells on a grey background. Some interior structure can be discerned in every cell (including the cell membrane, the dark grey cytoplasm, and the lighter cell nucleus with dark nucleoli inside). Cells that are close to division appear as bright, nearly circular objects. In a typical experiment images are captured concurrently for 600 of these ‘fields’. For each field we acquire about 900 images over a total duration of 48 hours, resulting in 260 GB of acquired data per experiment. The data processing task consists of segmenting each image and tracking all cells individually in time. The cloud application is designed to handle concurrent processing of many of these experiments and storing all input and output data in a structured way.

combination with miniaturized printed live cell assays. The acquisition system has a data rate of 1.5 MBps, and a typical 48 hour experiment can generate more than 260 GB of images, recorded as hundreds of multichannel videos each corresponding to a different treatment (Figure 1). Newer systems that we are currently evaluating can produce ten times more data in the same time.

The data analysis task for this platform is daunting: thousands of cells in the videos need to be tracked and characterized individually. The output consists of precise motion, morphological and gene expression data of each cell at many different timepoints. One of the particular systems we are studying is C2C12 myoblast differentiation - a process that results in the formation of muscle fibers from cultured cells. Cell differentiation, such as adipogenesis and myogenesis is mediated by gene transcription networks, which can be monitored by fluorescent probes. In a typical experiment, we need to measure the activity of several hundred different probes, called reporters, each of which records the activity of a gene regulatory circuit. For these experiments, the intensity of the fluorescent reporters is measured from the videos. While image analysis is the current bottleneck in our data processing pipeline, it happens to be a good candidate step for parallelization. The data processing can be broken up into hundreds of

independent video analysis tasks. The image analysis task uses computationally intensive code written in MATLAB to analyze both the data and generate result files. The analysis method we are currently developing solves the segmentation and tracking problem by first running a watershed segmentation algorithm. We then perform tracking by matching the segmented areas through time by using information about the cell shape intensity and position. As a final step we detect cell division events. The output data of the analysis is represented as a set of binary trees, each representing a separate cell lineage, where each node stores detailed information about a single cell at all time points.

To date we have used a local eight core server for data processing. A 600 video dataset takes up to 12h to process. This is the time required for one analysis of one experiment. Once the system will be fully operational, we will be acquiring large amounts of data (hundreds to thousands of GB per 48 hour experiment). We thus consider the development of a scalable and reliable cloud computing system for processing the data generated by our experiments of critical importance for our project.

2.2 Description of Specific Data Processing Problem

For each experiment (a specific set of parameters for the live cells under study), we may perform several data acquisitions. Typically, each acquisition generates 600 folders (one per field, see Figure 1), in which 900 acquired images are stored. Each image has a resolution of 512 x 512 16-bit pixels (512 KB), resulting in a total data size of 260 GB per acquisition. Different types of analysis (or data processing) jobs may be performed on the data gathered in each acquisition. Analysis jobs are normally performed using MATLAB programs, and the analysis can be parallelized easily, since each field can be processed independently. In the next Section we describe the design of a Hadoop-based cloud computing system for processing the data gathered in our live cell experiments.

3 System Design

3.1 Hadoop Components Used

As mentioned in Section I, we use Hadoop as our cloud computing framework. We use three Hadoop components: the MapReduce programming and execution environment, the reliable distributed file system called DFS, and a BigTable-like storage system for sparsely structured data called HBase. Each of the above components organizes cloud nodes into a master-slave structure. Figure 2 shows how the Hadoop components are used in our system. The DFS master node must be visible to all other components because they rely on DFS to store their data. Hadoop's MapReduce provides an API to write MapReduce programs as well as an environment to run those programs. Its master program is called "JobTracker", and slave programs are called "TaskTrackers". JobTracker accepts submitted user jobs, consisting of many map and reduce tasks with file-based

inputs. The large input files are split automatically into chunks that can be processed by the map workers (slaves) independently. The MapReduce framework distributes tasks with these input chunks to TaskTrackers to do the first-phase map computation. The results of the map phase are gathered and processed by the reduce phase, which generates final outputs. It is also acceptable to have map-only jobs. In this case, the reduce phase simply does nothing.

Hadoop's DFS is a flat-structure distributed file system. Its master node is called "namenode", and slave nodes are called "datanodes". Namenode is visible to all cloud nodes and provides a uniform global view for file paths in a traditional hierarchical structure. File contents are not stored hierarchically, but are divided into low level data chunks and stored in datanodes with replication. Data chunk pointers for files are linked to their corresponding locations by namenode.

HBase is a BigTable-like data store. It also employs a master-slave topology, where its master maintains a table-like view for users. Tables are split for distributed storage into row-wise "regions". The regions are stored on slave machines, using HBase's "region servers". Both the master and region servers rely on DFS to store data. Table I (See Figure 3) shows an example HBase table, which stores the web pages that link to web page www.cnn.com, along with the anchor text used in the link. Note that the order of the URL component strings is reversed in the row key, in an attempt to place links to pages that reside on the same web server in close-by rows, and thus making it likely that they are stored on the same datanode. The data stored in HBase are sorted key-value pairs logically organized as sparse tables indexed by row keys with corresponding column family values. Each column family represents one or more nested key-value pairs that are grouped together with the same column family as key prefix. The HBase table in Table I contains one row with key "com.cnn.www" and column family value "anchor:", which then contains two nested key-value pairs, with keys (web pages) "anchor:cnnsi.com" and "anchor:my.look.ca", and values (anchor text) "CNN" and "CNN.com".

3.2 System Overview

As shown in Figure 2, our system puts all essential functionality inside a cloud, while leaving only a simple Client at the experiment side for user interaction. In the cloud, we use DFS to store data, we use two HBase tables, called "Data" (Table II, Figure 3) and "Analysis" (Table III, Figure 3), to store metadata for data and for analysis jobs, respectively, and we use MapReduce to do computation. The Client can issue three types of simple requests to the cloud application (via the Cloud Frontend): a request for transferring experiment data (an acquisition) into the cloud's DFS, a request for performing an analysis job on a certain acquisition using a certain analysis program, and a request for querying/viewing analysis results. The Cloud Frontend processes Client requests. When it receives a data-transfer request, it starts an scp connection to the Client's local storage, and transfers data to its Temporary Staging Storage. It then puts the staged data from the Temporary Staging Storage into the DFS. It also updates the "Data" table in HBase to record the metadata that describes the acquisition (including

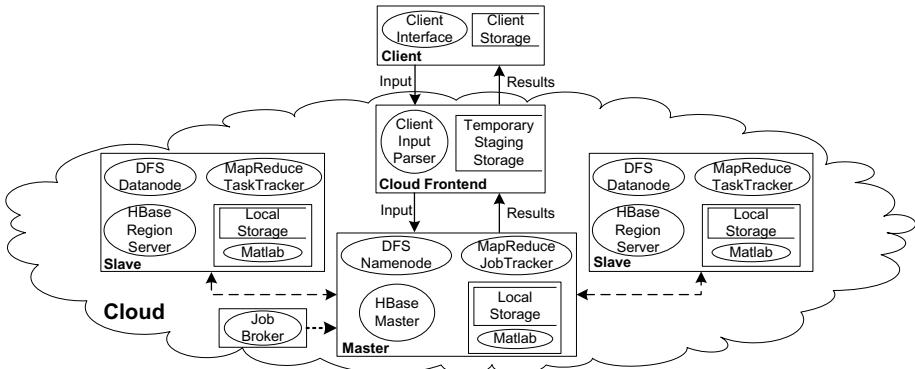


Fig. 2. System design diagram, showing the Hadoop components and cloud application components

TABLE I
EXAMPLE HBASE TABLE

Row Key	Column Family "anchor:"	
com.cnn.www	anchor:cnnsi.com	CNN
	anchor:my.look.ca	CNN.com

TABLE II
DATA TABLE IN HBASE

Row Key	Type	User	Acq ID	Exp. ID	Column Family: Raw:		Column Family: Result:	
					Raw: FieldRange	1-10		
DataKey1	Raw	X	1	A	Raw: FileParentDir	Data/expA/acq1/raw/		
DataKey2	Result	Y	1	A		Result:FieldRange	1-2	
						Result:FileParentDir	Data/expA/acq1/Result/round1	
						Result:RoundNum	1	

TABLE III
ANALYSIS TABLE IN HBASE

Row Key	User	Input Data Row Key	Output Data Row Key	Round Number	Column Family: Program:		Status
AnalysisKey1	Y	DataKey1	DataKey2	1	Program:Name	CellLineage	new
					Program:Version	1	
					Program:Option	4	
					Program:InputFieldRange	1-2	

Fig. 3. HBase tables

the range of fields recorded in the acquisition, and the DFS path to the folder containing the fields). In the “Data” table, we distinguish records by type. The “Raw” type denotes raw experiment data, and the properties of a raw data acquisition are stored in the column family “Raw”. The “Result” type denotes result data, which has different parameters than raw data, which are thus stored in a

different column family. In HBase's sparse data implementation, empty column families are not actually stored and do not take up resources; they are shown as empty in the logical table view maintained by the HBase Master. If the request is job submission or query, it inserts a record into the "Analysis" table in HBase or queries the "Analysis" table for the required information. The Job Broker shown in the bottom left corner of Figure 2 polls the "Analysis" table through regular "heart-beat" intervals to discover newly inserted unprocessed jobs, and submits the MapReduce jobs to the Master node Job Tracker to get processing started. The Job Broker can be run on any node of the Cloud. The Master node contains the DFS namenode, the MapReduce Job Tracker and the HBase master. We put the master processes of all three Hadoop components on one single Master node for simplicity, while they may also be put on separate nodes. All other nodes run as slaves to do both data storage and task processing. In our case, MATLAB is a native program that cannot use DFS files directly but requires its input and output files to reside on the local file system, we need to get files out of the DFS and copy them to local storage before MATLAB can start processing. Local storage, together with the MATLAB process started by the MapReduce application program, is shown as an isolated box inside each node in Figure 2. After MATLAB processing completes, the results are put back into DFS, and, when all Map tasks have been completed, the "Analysis" table in HBase is updated accordingly to reflect the change of status from "new" to "complete".

3.3 Programming MapReduce

In order to use Hadoop for our problem, it is necessary to extend the default way how Hadoop handles input data formats, how it handles the way input data are split into parts for processing by the map workers, and how it handles the extraction of atomic data records from the split data. Hadoop normally processes a very large data file containing a long sequence of atomic input records that each can be processed independently. The file is split automatically, normally along DFS block boundaries, and, due to the granularity of the atomic input records, the input splits usually each contain many atomic data records. For example, the atomic input record may be a line of text in a file separated by carriage returns. The file can be split at arbitrary locations, and record boundaries can be recovered easily. This type of automatic splitting is efficient and convenient for this type of problems.

For our scientific data processing application, however, the situation is different. In our case, an atomic data record is a folder of images corresponding to one field (total data size 512KB x number of images in that folder). The granularity is much coarser, and when we split the input, we may require just a few atomic input records per split (i.e., per map worker). In this case, it is more convenient and efficient to let the user control the number of splits, and to perform the split exactly at the boundary of the atomic input records. Also, it is more natural to provide the input indirectly, via paths to the DFS folders that contain the image

files, rather than providing the complete data set serialized into a single large file for all map tasks.

We have implemented this approach by writing new classes that implement the Hadoop interfaces for handling input and input splits. We have implemented the following classes:

1. `StringArrayInputFormat.java` (implements Hadoop's `InputFormat` interface).
2. `CommaSeparatedStringInputSplit.java` (implements Hadoop's `InputSplit` interface).
3. `CommaSeparatedStringInputSplitRecordReader.java` (implements Hadoop's `RecordReader` interface).

`StringArrayInputFormat` accepts as input a string array that contains the DFS paths to all the field data folders, and gets the number of splits that the user application desires from job configuration. It splits the input into smaller string array objects defined by `CommaSeparatedStringInputSplit.java`. When a map task is executed on a slave node, it is passed one of these small string arrays, specifying the input folders for the split to be processed. It then calls `CommaSeparatedStringRecordReader` to extract atomic input records (which each point to one field folder). The reader can download simplified versions of these classes from our project website [27].

In the next section we use this implementation for our cell data processing case study and evaluate its performance. Note that this approach can also be used to make Hadoop suitable for many other types of problems that do not directly fit within Hadoop's default context, namely, processing very long sequences of small atomic input records serialized into a single big file that is split automatically. In the discussion section below we give the example of processing records directly from a database table that is accessed remotely.

4 Performance Analysis and Optimizations

4.1 Environment Setup

We use a homogeneous cluster to do initial system development and proof-of-concept tests. The cluster is comprised of 21 SunFire X4100 servers with two dual-core AMD Opteron 280 CPUs interconnected by Gigabit Ethernet. The tests described below are performed on a smaller set of data, in which each 'field' folder contains 300 compressed images of size 300KB, for a total data size of 90MB per folder. The cloud is composed of 20 slave nodes for all tests.

4.2 MapReduce Performance

In this section we describe some simple runtime tests with our Hadoop-based cloud computing framework, mainly to show the functionality of our solution, and to give some insight in its performance. In fact, the use of hadoop allows to speed up calculations by a factor that equals the number of worker nodes,

except for startup effects, which are relatively small when the execution time of individual tasks is large enough.

Figure 4 shows a test in which we run an increasing number of tasks with one folder per task. It can be seen that the total time increases, which means that there is an overhead for scheduling.

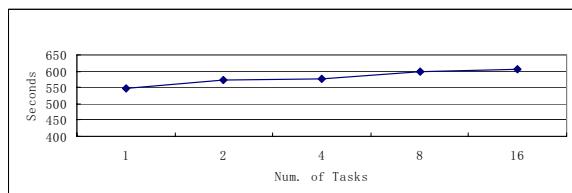


Fig. 4. Test for scheduling overhead

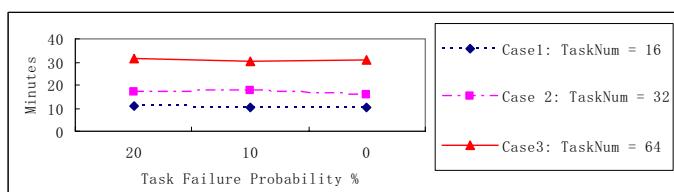


Fig. 5. Test for the degree of fault tolerance

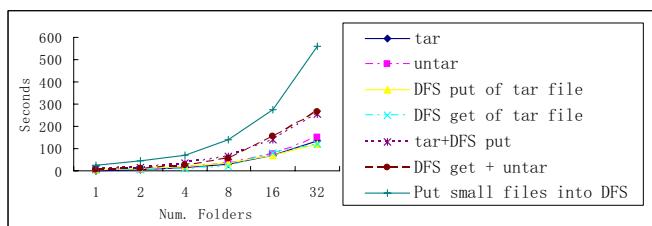


Fig. 6. Tar and DFS Performance

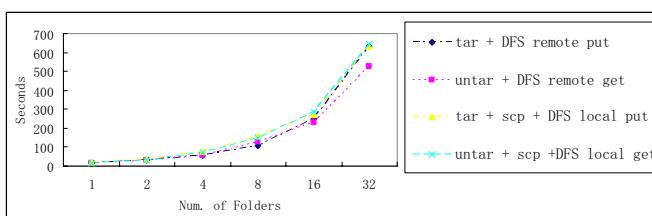


Fig. 7. DFS as remote staging tool

Figure 5 shows how the system is resilient against failures. We again consider jobs with one folder per task. Each task has a failure rate of 0%, 10% or 20%. (We artificially stop some of the tasks right after they start, with the probabilities specified.) Figure 5 shows that Hadoop is able to recover from these failures as expected. (It restarts tasks that have failed automatically.) The total time does not increase markedly when tasks fail, due to the fact that we let tasks fail immediately after they start. This shows that there is not much overhead associated with the fault-tolerance mechanism. The figure also shows that 64 tasks take about twice the time of 32 tasks, as expected. Interestingly, 32 tasks take less than double the time of 16 tasks (on 20 cloud nodes), which is likely due to an overhead cost associated with starting and completing a job.

4.3 Remote Data Staging and DFS Performance

In this section, we discuss optimizations in data transfer. It turns out that using tar files per directory results in a much more efficient system, both for transfer of data to and from the cloud, and for putting data from temporary storage into DFS and back.

In our system, data circulate through the following cycle:

1. Remote staging-in of raw experiment data: From Client to Cloud Front End.
2. Put raw data into DFS: From Cloud Front End to DFS.
3. Get raw data from DFS for MATLAB to process at each node: From DFS to Local Storage.
4. Put result data back to DFS after processing at each node: From Local Storage to DFS.
5. Get result data from DFS: From DFS to Cloud Front End.
6. Remote staging-out of result data: From Cloud Front End to Client.

Figure 6 shows that it takes approximately the same time to tar and untar a data folder, as it takes to put the tared file into DFS or take it out. Putting the untared folder into DFS takes unacceptably long. Therefore, we use tared files in our implementation.

Figure 7 shows that DFS can also be used efficiently as a remote data-staging tool: rather than first performing an scp transfer from the client to the cloud, followed by putting the tared folder from the cloud temporary storage into DFS, one can also run a Hadoop DFS datanode on the client machine outside the cloud, and put the data directly into DFS. Figure 7 shows that these two methods have almost the same efficiency. However, due to firewall constraints this may not be easy to do across organizations.

5 Discussion

5.1 Advantages and Disadvantages of Using HBase

One could consider using a regular relational database instead of HBase. Compared to a database, one major drawback of HBase is its inability to handle

complex queries. This is because HBase organizes data in sets containing key-value pairs, rather than relations. The query model used in relational databases can thus not be applied to HBase. We don't use a relational database in our project because of the following reasons:

1. It is complex and expensive to install and deploy large-scale scalable database management systems.
2. For our application, we don't need complex database-like queries as supported by database systems, and HBase performs well for the tasks we need it for.
3. DFS provides reliable storage, and current database systems cannot make use of DFS directly. Therefore, HBase may have a better degree of fault tolerance for large-scale data management in some sense.
4. Most importantly, we find it natural and convenient to model our data using sparse tables because we have varying numbers of fields with different metadata, and the table organization can change significantly as the usage of the system is extended to new analysis programs and new types of input or output data.

Because of these reasons, we believe that HBase is a good choice to do metadata management for problems like ours.

5.2 Extending Our Approach to Other Types of Data Processing

It is easy to extend our approach to other types of data processing based on our classes. For example, one can use a database table as input for doing simple distributed query processing using Hadoop's default approach, by retrieving all the table contents, putting it into one serialized file as input, and implementing a record reader that can detect atomic input record boundaries correctly. However, it is much more natural to let the Map workers read their own set of records from the database table directly. To this end, users can write a new class which extends our `StringArrayInputFormat.java` class, and override our `getSplits` method in it. It is easy to pass in the database `jdbc url` and the number of splits desired. The `getSplits` method would then first determine the total number of rows in the table, and then split the table into row ranges according to the specified number of splits. Each row range is then written in our `CommaSeperatedStringInputSplit` format as a split. Then the user can implement a record reader class such that, in the Mapper program, the database table will be accessed by the `jdbc url` and the row range specified in each split will be retrieved by the record reader and processed by the Mapper. Example code for remote database access can be downloaded from our website [27].

6 Related Work

Cloud computing and its usage are still in their infancy and not much literature is available in the forum of academic publications. As we've mentioned,

virtualization is used to construct a homogeneous environment for clouds while homogeneous clusters can also be easily configured for this purpose. Considering virtualization, [21] studies a negotiation and leasing framework for Cluster on Demand built from both Xen [7] virtual machines and physical machines. They built a Grid hosting environment as in [26] while a similar platform with simple unified scheduling among homogeneous machines can make this a Cloud environment. The authors of [28] study automatic virtual machine configuration for database workloads, in which several database management system instances, each running in a virtual machine, are sharing a common pool of physical computing resources. The performance optimization they made by controlling the configurations of the virtual machines in which they run may be extended to cloud environments for automatic configuration. Google File System [18] with its open source implementation Hadoop DFS is currently a popular choice for a cloud file system. BigTable [11] and its open source implementation HBase act as data organization tools on top of the underlying distributed file system in the form of distributed column-oriented datastores. Other scalable data hosting and organization solutions include Amazon's Dynamo, Yahoo's PNUTS, Sinfonia, etc. Various data processing languages to facilitate usage of distributed data structures are also investigated in Sawzall [25], Pig Latin [24], and SCOPE [10]. Google's MapReduce [13], Microsoft's Dryad [22], and Clustera [15] investigate distributed programming and execution frameworks. MapReduce aims at simplicity with limited generality while Dryad provides generality but is complex to write programs with. Clustera is similar to Dryad but uses a different scheduling mechanism. MapReduce has been used for a variety of applications [9][17][23].

7 Conclusions and Future Work

In this paper, we have evaluated Hadoop by doing a case study on a scientific data processing problem with MATLAB. In the case study, we extend the capability of Hadoop's MapReduce in accepting arbitrary input formats with explicit split control. Our system design and implementation can be reused for similar application scenarios and extended to arbitrary applications with divisible inputs. We have discussed performance evaluation runs and optimizations in terms of data transfer mechanisms.

The scientific data-processing problem considered in this paper is simple: the workload is divisible, without the need for communication between tasks. Nevertheless, our cloud solution is attractive because it takes advantage of many of the desirable features offered by the cloud concept and Hadoop, including scalability, reliability, fault-tolerance, easy deployability, etc. Our approach can be generalized easily to more complicated scientific data processing jobs than the examples given above. For instance, we have already designed and implemented a computational workflow system based on Hadoop that supports workflows built from MapReduce programs as well as legacy programs (existing programs not built by using the MapReduce API) [30]. We also plan to investigate how we can process jobs with intertask communications. Building on the experience

presented in this paper, we will address Hadoop-based cloud computing for more complicated scientific computing applications in future work.

Acknowledgements

We want to give thanks to Genome Quebec, the University of Waterloo Faculty of Mathematics and David R. Cheriton School of Computer Science, and SHARCNET for providing the computing infrastructure for this work.

References

1. Aguilera, M.K., Merchant, A., Shah, M.A., Veitch, A.C., Karamanolis, C.T.: Sinfonia: A New Paradigm for Building Scalable Distributed Systems. In: SOSP 2007 (2007)
2. Aguilera, M., Golab, W., Shah, M.: A Practical Scalable Distributed B-Tree. In: VLDB 2008 (2008)
3. Amazon Elastic Compute Cloud, <http://aws.amazon.com/ec2/> (retrieved date: September 27, 2009)
4. Apache Hadoop, <http://hadoop.apache.org/> (retrieved date: September 27, 2009)
5. Apache HBase, <http://hadoop.apache.org/hbase/> (retrieved date: September 27, 2009)
6. Apache Hama, <http://incubator.apache.org/hama/> (retrieved date: September 27, 2000)
7. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: SOSP 2003 (2003)
8. Brantner, M., Florescu, D., Graf, D.A., Kossman, D., Kraska, T.: Building a Database on S3. In: SIGMOD 2008 (2008)
9. Catanzaro, B., Sundaram, N., Keutzer, K.: A MapReduce framework for programming graphics processors. In: Workshop on Software Tools for MultiCore Systems (2008)
10. Chaiken, R., Jenkins, B., Larson, P., Ramsey, B., Shakib, D., Weaver, S., Zhou, J.: SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. In: VLDB 2008 (2008)
11. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.: BigTable: A Distributed Storage System for Structured Data. In: OSDI 2006 (2006)
12. Cooper, B., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D., Yerneni, R.: PNUTS: Yahoo!'s Hosted Data Serving Platform. In: VLDB 2008 (2008)
13. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI 2004 (2004)
14. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's Highly Available Key-Value Store. In: SOSP 2007 (2007)
15. DeWitt, D.J., Robinson, E., Shankar, S., Paulson, E., Naughton, J., Krioukov, A., Royalty, J.: ClusterA: An Integrated Computation and Data Management System. In: VLDB 2008 (2008)

16. ELASTRA, <http://www.elastracom/> (retrieved date: Sepember 27, 2009)
17. Elsayed, T., Lin, J., Oard, D.: Pairwise Document Similarity in Large Collections with MapReduce. In: Proc. Annual Meeting of the Association for Computational Linguistics (2008)
18. Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google File System. In: SOSP 2003 (2003)
19. GigaSpaces, <http://www.gigaspaces.com/> (retrieved date: September 27, 2009)
20. Google and IBM Announce University Initiative, <http://www.ibm.com/ibm/ideasfromibm/us/google/index.shtml> (retrieved date: September 27, 2009)
21. Irwin, D.E., Chase, J.S., Grit, L.E., Yumerefendi, A.R., Becker, D., Yocum, K.: Sharing Networked Resources with Brokered Leases. In: USENIX Annual Conference 2006 (2006)
22. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In: EuroSys 2007 (2007)
23. McNabb, A.W., Monson, C.K., Seppi, K.D.: MRPSO: MapReduce Particle Swarm Optimization. In: Genetic and Evolutionary Computation Conference (2007)
24. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig Latin: A Not-So-Foreign Language for Data Processing. In: SIGMOD 2008 (2008)
25. Pike, R., Dorward, S., Griesemer, R., Quinlan, S.: Interpreting the Data: Parallel Analysis with Sawzall. Scientific Programming 13(4) (2005)
26. Ramakrishnan, L., Irwin, D.E., Grit, L.E., Yumerefendi, A.R., Iamnitchi, A., Chase, J.S.: Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control. In: SC 2006 (2006)
27. Scalable Scientific Computing Group, University of Waterloo, <http://www.math.uwaterloo.ca/groups/SSC/software/cloud> (retrieved date: September 27, 2009)
28. Soror, A., Minhas, U.F., Aboulnaga, A., Salem, K., Kokosieliis, P., Kamath, S.: Automatic Virtual Machine Configuration for Database Workloads. In: SIGMOD 2008 (2008)
29. Yang, H.C., Dasdan, A., Hsiao, R.-L., Parker, D.S.: Map-reduce-merge: simplified relational data processing on large clusters. In: SIGMOD 2007 (2007)
30. Zhang, C., De Sterck, H.: CloudWF: A Computational Work ow System for Clouds Based on Hadoop. In: The First International Conference on Cloud Computing, Beijing, China (2009)

Author Index

- Aboulnaga, Ashraf 400
Agarwal, Ashok 283
Altenfeld, Ralph 62
an Mey, Dieter 62, 366
Armstrong, Patrick 283
- Bandrauk, André Dieter 70, 99, 134
Bauer, Michael A. 230, 241
Behdinan, Kamran 148
Bespalko, Dustin 1
Blight, Wilfred 269
Bourgault, Yves 30
Bücker, H. Martin 48
Bungay, Sharene D. 196
- Cann, Natalie M. 76
Carrington Jr., Tucker 109
Chang, Dongil 20
Charbonneau, Andre 283
Chen, Hao 283
Choudhary, Alok 304
- Dantas, Mario A.R. 230, 241
da Silva, Alexandre P.C. 241
Desmarais, Ronald J. 283
De Sterck, Hans 337, 400
Djambazian, Haig 400
Doyle, Matthew G. 30
- Eberl, Hermann J. 180
Fawaz, Zouheir 148
Filizadeh, Shaahin 165
Fortmeier, Oliver 48
- Gable, Ian 283
Gole, Ani 165
Goodenough, David G. 283
Graham, Peter 165
Guan, Aimin 283
- Hurley, Richard 269
- Impey, Roger 283
Iwainsky, Christian 62
- Janson, Denise 241
Jia, Zongchao 117
Kapinos, Paul 366
Kwan, Peter 117
- Layton, Jeffrey 292
Lorin, Emmanuel 70
Lu, HuiZhong 99
- McConnell, Sabine 269
Mendonça, R.P. 230
Meng, Xiaoxuan 349
Merkulow, Igor 366
Moa, Belaid 283
Montesano, John 148
Moreno Maza, Marc 378
Muhammad, Nasim 180
- Nakka, Nithin 304
- Patton, David 269
Penka Fowe, Emmanuel 134
Podaima, Wayne 283
Poirier, Raymond A. 196
Pollard, Andrew 1
Poon, Cheung 148
Pourreza, Hossein 165
- Qin, Jinhui 241
- Reinefeld, Alexander 323
Rostrup, Scott 337
- Sarholz, Samuel 62
Sarkar, Abhijit 251
Schleiden, Christopher 366
Schütt, Thorsten 323
Si, Chengxiang 349
Sladek, Rob 400
Sobie, Randall 283
Staveley, Mark S. 196
Subber, Waad 251
Sykes, Edward R. 215
- Tavoularis, Stavros 20, 30
Terboven, Christian 366

- Uddin, Mesbah 1
Viegas, Diogo R. 230
Walker, Virginia K. 117
Wang, Shihao 76
Wang, Xiao-Gang 109
Wathen, Brent 117
Xie, Yuzhen 378
Xu, Lu 349
Young, Graeme 269
Zhang, Chen 400