

LNCS 7640

Ioannis Caragiannis et al. (Eds.)

Euro-Par 2012: Parallel Processing Workshops

**BDMC, CGWS, HeteroPar, HiBB, OMHI,
Paraphrase, PROPER, Resilience, UCHPC, VHPC**
Rhodes Island, Greece, August 2012
Revised Selected Papers



Springer

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 for Informatics, Saarbruecken, Germany

Ioannis Caragiannis Michael Alexander
Rosa Maria Badia Mario Cannataro
Alexandru Costan Marco Danelutto
Frédéric Desprez Bettina Krammer
Julio Sahuquillo Stephen L. Scott
Josef Weidendorfer (Eds.)

Euro-Par 2012: Parallel Processing Workshops

BDMC, CGWS, HeteroPar, HiBB, OMHI,
Paraphrase, PROPER, Resilience, UCHPC, VHPC
Rhodes Island, Greece, August 27-31, 2012
Revised Selected Papers

Volume Editors

Ioannis Caragiannis; E-mail: caragian@ceid.upatras.gr

Michael Alexander; E-mail: malexand@scilytics.com

Rosa Maria Badia; E-mail: rosa.m.badia@bsc.es

Mario Cannataro; E-mail: cannataro@unicz.it

Alexandru Costan; E-mail: alexandru.costan@inria.fr

Marco Danelutto; E-mail: marcod@di.unipi.it

Frédéric Desprez; E-mail: frederic.desprez@inria.fr

Bettina Krammer; E-mail: bettina.krammer@uvsq.fr

Julio Sahuquillo; E-mail: jsahuqui@disca.upv.es

Stephen L. Scott; E-mail: sscott@tntech.edu

Josef Weidendorfer, E-mail: weidendo@in.tum.de

ISSN 0302-9743

e-ISSN 1611-3349

ISBN 978-3-642-36948-3

e-ISBN 978-3-642-36949-0

DOI 10.1007/978-3-642-36949-0

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013932224

CR Subject Classification (1998): C.4, C.2.4, H.3.4, C.1, D.4.1-3, D.4.7, C.3, B.8.2, H.2.4, F.1.2, F.2.2, D.2.2-3, I.3.1, J.3

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

© Springer-Verlag Berlin Heidelberg 2013

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.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Euro-Par is an annual series of international conferences dedicated to the promotion and advancement of all aspects of parallel and distributed computing. Euro-Par 2012 was the 18th edition in this conference series. Euro-Par covers a wide spectrum of topics from algorithms and theory to software technology and hardware-related issues, with application areas ranging from scientific to mobile and cloud computing. Euro-Par provides a forum for the introduction, presentation, and discussion of the latest scientific and technical advances, extending the frontier of both the state of the art and the state of the practice.

Since 2006, Euro-Par conferences have provided a platform for a number of accompanying, technical workshops. This is a great opportunity for small and emerging communities to meet and discuss focussed research topics.

For the 2012 edition, a more formal management procedure for Euro-Par Workshops was initiated. A Workshop Chairpersonship was created, with two Co-chairs: one from the Organizing Team (Ioannis Caragiannis), and one from the Euro-Par Steering Committee (Luc Bougé). The rationale behind this choice has been to achieve a strong connection with the hosting team as in previous editions and furthermore maintain a continuity along the years for the interaction with the workshop organizers. In the coordination process, we were assisted by two historical Euro-Par workshop organizers, Michael Alexander and Jesper Träff; we would like to warmly thank them for their help.

A Call for Workshop Proposals was issued, and the proposals were reviewed by the Co-chairs, with advice by the Euro-Par Steering Committee members and (in some cases) the Workshop Advisory Board. Overall, the following ten workshops were selected for the 2012 edition. The list includes seven workshops that had been collocated with Euro-Par in the previous years as well as three newcomers.

- First Workshop on Big Data Management in Clouds (BDMC)
- CoreGRID/ERCIM Workshop on Grids, Clouds, and P2P Computing (CGWS)
- 10th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar)
- Third Workshop on High-Performance Bioinformatics and Biomedicine (HiBB)
- First Workshop on On-chip Memory Hierarchies and Interconnects: Organization, Management, and Implementation (OMHI)
- Paraphrase Workshop 2012
- 5th Workshop on Productivity and Performance (PROPER)
- Workshop on Resiliency in High-Performance Computing (Resilience)
- 5th Workshop on UnConventional High-Performance Computing (UCHPC)
- 7th Workshop on Virtualization in High-Performance Cloud Computing (VHPC)

After the selection of these workshops, we enforced a synchronization of workshop submission deadlines, notification dates, and final/revised version submission deadlines that were followed by all workshops. We hope that the new procedures established with the 2012 edition will be adopted and further enhanced in the future.

The present volume includes the proceedings of all workshops. Each workshop had their own paper-reviewing process. Special thanks are due to the authors of all the submitted papers, the members of the Program Committees, all the reviewers and the workshop organizers. The workshops were successful because of their hard work.

We are also grateful to the Euro-Par General Chairs and the members of the Euro-Par Steering Committee – in particular to Christos Kaklamanis and Christian Lengauer – for their support and advice regarding the coordination of workshops. Furthermore, we thank Raymond Namyst, the Workshop Chair of Euro-Par 2011, for sharing his experience with us.

It was our pleasure and honor to organize and host the Euro-Par 2012 workshops in Rhodes. We hope all the participants enjoyed the technical program and the social events organized during the conference.

November 2012

Ioannis Caragiannis
Luc Bougé

Organization

Euro-Par Steering Committee

Chair

Chris Lengauer University of Passau, Germany

Vice-Chair

Luc Bougé ENS Cachan, France

European Representatives

José Cunha	New University of Lisbon, Portugal
Marco Danelutto	University of Pisa, Italy
Emmanuel Jeannot	LaBRI-INRIA, France
Christos Kaklamanis	Computer Technology Institute and Press “Diophantus”, Greece
Paul Kelly	Imperial College, UK
Thomas Ludwig	University of Hamburg, Germany
Emilio Luque	Autonomous University of Barcelona, Spain
Tomàs Margalef	Autonomous University of Barcelona, Spain
Wolfgang Nagel	Dresden University of Technology, Germany
Rizos Sakellariou	University of Manchester, UK
Henk Sips	Delft University of Technology, The Netherlands
Domenico Talia	University of Calabria, Italy

Honorary Members

Ron Perrott	Queen's University Belfast, UK
Karl Dieter Reinartz	University of Erlangen-Nuremberg, Germany

Observer

Felix Wolf RWTH Aachen, Germany

Euro-Par 2012 Organization

Conference Co-chairs

Christos Kaklamanis	CTI and University of Patras, Greece
Theodore Papatheodorou	University of Patras, Greece
Paul Spirakis	CTI and University of Patras, Greece

VIII Organization

Workshop Co-chairs

Luc Bougé

Ioannis Caragiannis

ENS Cachan, France

CTI and University of Patras, Greece

Local Organizing Committee

Katerina Antonopoulou

CTI, Greece

Stavros Athanassopoulos

CTI and University of Patras, Greece

Rozina Efstathiadou

CTI, Greece

Lena Gourdoupi

CTI, Greece

Panagiotis Kanellopoulos

CTI and University of Patras, Greece

Evi Papaioannou

CTI and University of Patras, Greece

Euro-Par 2012 Workshops

Workshop Co-chairs

Luc Bougé

ENS Cachan, France

Ioannis Caragiannis

CTI and University of Patras, Greece

First Workshop on Big Data Management in Clouds (BDMC 2012)

Program Chairs

Alexandru Costan

INRIA Rennes - Bretagne Atlantique, France

Ciprian Dobre

University Politehnica of Bucharest, Romania

Program Committee

Gabriel Antoniu

INRIA, France

Luc Bougé

ENS, France

Toni Cortes

Universitat Politecnica de Catalunya, Spain

Valentin Cristea

University Politehnica Bucharest, Romania

Frédéric Desprez

INRIA, France

David Menga

EDF, France

Maria S. Perez

Universidad Politecnica de Madrid, Spain

Guillaume Pierre

VU University, The Netherlands

Judy Qiu

Indiana University, USA

Leonardo Querzoni

University of Rome, La Sapienza, Italy

Domenico Talia

University of Calabria, Italy

Osamu Tatebe

University of Tsukuba, Japan

CoreGRID/ERCIM Workshop on Grids, Clouds, and P2P Computing (CGWS 2012)

Program Chairs

Frédéric Desprez	INRIA and ENS Lyon, France
Domenico Talia	University of Calabria, Italy
Ramin Yahyapour	University of Göttingen, Germany

Program Committee

Marco Aldinucci	University of Turin, Italy
Alvaro Arenas	IE Business School, Madrid, Spain
Augusto Ciuffoletti	University of Pisa, Italy
Marco Danelutto	University of Pisa, Italy
Frédéric Desprez	INRIA and ENS Lyon, France
Paraskevi Fragopoulou	FORTH-ICS, Greece
Vladimir Getov	University of Westminster, UK
Sergei Gorlatch	University of Münster, Germany
Philippe Massonet	CETIC, Belgium
Carlo Mastroianni	ICAR-CNR, Italy
Omer Rana	Cardiff University, UK
Rizos Sakellariou	University of Manchester, UK
Alan Stewart	Queen's University of Belfast, UK
Domenico Talia	University of Calabria, Italy
Ramin Yahyapour	University of Göttingen, Germany
Wolfgang Ziegler	Fraunhofer Institute SCAI, Germany

10th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar 2012)

Program Chair

Rosa Badia	Barcelona Supercomputing Center, Spain
------------	--

Steering Committee

Domingo Giménez	University of Murcia, Spain
Alexey Kalinov	Cadence Design Systems, Russia
Alexey Lastovetsky	University College Dublin, Ireland
Yves Robert	Ecole Normale Supérieure de Lyon, France
Leonel Sousa	INESC-ID/IST, Technical University of Lisbon, Portugal
Denis Trystram	LIG, Grenoble, France

Program Committee

Jacques Bahi	University of Franche-Comte, France
Jorge Barbosa	FEUP, Portugal
Olivier Beaumont	INRIA Futurs Bordeaux, LABRI, France
François Bodin	CAPS, France
Cristina Boeres	Universidade Federal Fluminense, Brazil
George Bosilca	Innovative Computing Laboratory - University of Tennessee, Knoxville, USA
Alejandro Duran	Intel, Spain
Pierre-Francois Dutot	ID-IMAG, France
Toshio Endo	Tokyo Institute of Technology, Japan
Edgar Gabriel	University of Houston, USA
Shuichi Ichikawa	Toyohashi University of Technology, Japan
Emmanuel Jeannot	LORIA - INRIA Nancy Grand-Est, France
Helen Karatza	Aristotle University of Thessaloniki, Greece
Hatem Ltaief	KAUST, Saudi Arabia
Pierre Manneback	University of Mons, Belgium
Satoshi Matsuoka	Tokyo Institute of Technology, Japan
Wahid Nasri	Higher School of Sciences and Techniques of Tunis, Tunisia
Nacho Navarro	Barcelona Supercomputing Center, Spain
Dana Petcu	West University of Timisoara, Romania
Serge Petiton	Université des Sciences et Technologies de Lille, France
Antonio J. Plaza	University of Extremadura, Spain
Enrique Quintana	Universidad Jaime I, Spain
Vladimir Rychkov	University College Dublin, Ireland
Mitsuhisa Sato	University of Tsukuba, Japan
Howard Jay Siegel	Colorado State University, USA
Leonel Sousa	INESC-ID/IST, Technical University of Lisbon, Portugal
Antonio M. Vidal	Universidad Politecnica de Valencia, Spain

Third Workshop on High-Performance Bioinformatics and Biomedicine (HiBB 2012)

Program Chair

Mario Cannataro	University Magna Græcia of Catanzaro, Italy
-----------------	---

Program Committee

Pratul K. Agarwal	Oak Ridge National Laboratory, USA
David A. Bader	College of Computing, Georgia University of Technology, USA

Ignacio Blanquer	Universidad Politecnica de Valencia, Spain
Daniela Calvetti	Case Western Reserve University, USA
Werner Dubitzky	University of Ulster, UK
Ananth Y. Grama	Purdue University, USA
Concettina Guerra	Georgia Institute of Technology, USA
Vicente Hernandez	Universidad Politecnica de Valencia, Spain
Marcelo Lobosco	Federal University of Juiz de Fora, Brazil
Salvatore Orlando	University of Venice, Italy
Omer F. Rana	Cardiff University, UK
Richard Sinnott	University of Melbourne, Australia
Fabrizio Silvestri	ISTI-CNR, Italy
Erkki Somersalo	Case Western Reserve University, USA
Paolo Trunfio	University of Calabria, Italy
Albert Zomaya	University of Sydney, Australia

First Workshop on On-chip Memory Hierarchies and Interconnects: organization, management, and implementation (OMHI 2012)

Program Chairs

Julio Sahuquillo	Universitat Politècnica de València, Spain
Maria Engracia Gómez	Universitat Politècnica de València, Spain
Salvador Petit	Universitat Politècnica de València, Spain

Program Committee

Marcello Coppola	STMicroelectronics, France
Giorgos Dimitrakopoulos	Democritus University of Thrace, Greece
Pierfrancesco Foglia	Università di Pisa, Italy
Crispín Gómez	Intel labs, Barcelona, Spain
Kees Goossens	Eindhoven University of Technology, The Netherlands
David Kaeli	Northeastern University, USA
Stefanos Kaxiras	Uppsala University, Sweden
Sonia López	Rochester Institute of Technology, USA
Iakovos Mavroidis	Foundation for Research and Technology - Hellas (FORTH), Greece
Orlando Moreira	ST-Ericsson, The Netherlands
Federico Silla	Universitat Politècnica de València, Spain
Tor Skeie	Simula Research Laboratory, Norway
Rafael Ubal	Northeastern University, USA

Paraphrase Workshop 2012

Program Chairs

Marco Danelutto	University of Pisa, Italy
Kevin Hammond	University of St. Andrews, UK
Horacio Gonzalez-Velez	Robert Gordon University, Aberdeen, UK

Program Committee

Marco Aldinucci	University of Turin, Italy
Francesco Cesarini	Erlang Solutions Ltd London, UK
Shoner Holger	Software Competence Center in Hagenberg, Austria
Colin Glass	University of Stuttgart, Germany
Sergei Gorlatch	University of Münster, Germany
Peter Kilpatrick	Computer Science Queen's University Belfast, UK
Edward Tate	Erlang Solutions Ltd London, UK

5th Workshop on Productivity and Performance (PROPER 2012)

Program Chair

Andreas Knüpfer	TU Dresden, Germany
-----------------	---------------------

Steering Committee

Michael Gerndt	TU München, Germany
Bettina Krammer	Université de Versailles St-Quentin-en-Yvelines, France
Shirley Moore	University of Tennessee, USA
Matthias Müller	TU Dresden, Germany
Felix Wolf	German Research School for Simulation Sciences, Germany

Program Committee

Jean-Thomas Acquaviva	Exascale Computing Research, France
Dieter an Mey	RWTH Aachen, Germany
Patrick Carribault	CEA, France
Jens Doleschal	TU Dresden, Germany
Karl Fürlinger	University of California at Berkeley, USA
Michael Gerndt	TU München, Germany
Andreas Knüpfer	TU Dresden, Germany

Bettina Krammer	Université de Versailles St-Quentin-en-Yvelines, France
Allen Malony	University of Oregon, USA
Matthias Müller	TU Dresden, Germany
Shirley Moore	University of Tennessee, USA
Marc Pérache	CEA, France
Martin Schulz	Lawrence Livermore National Lab, USA
Jan Treibig	RRZE, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
Felix Wolf	German Research School for Simulation Sciences, Germany

Workshop on Resiliency in High-Performance Computing (Resilience 2012)

Program Chairs

Stephen L. Scott	Oak Ridge National Laboratory, USA
Chokchai (Box) Leangsuksun	Louisiana Tech University, USA
Christian Engelmann	Oak Ridge National Laboratory, USA

Program Committee

Vassil Alexandrov	Barcelona Supercomputing Center, Spain
David E. Bernholdt	Oak Ridge National Laboratory, USA
George Bosilca	University of Tennessee, USA
Jim Brandt	Sandia National Laboratories, USA
Patrick G. Bridges	University of New Mexico, USA
Greg Bronevetsky	Lawrence Livermore National Laboratory, USA
Franck Cappello	INRIA/UIUC, France/USA
Zizhong Chen	Colorado School of Mines, USA
Nathan DeBardeleben	Los Alamos National Laboratory, USA
Christian Engelmann	Oak Ridge National Laboratory, USA
Jack Dongarra	University of Tennessee, USA
Christian Engelmann	Oak Ridge National Laboratory, USA
Yung-Chin Fang	Dell, USA
Kurt B. Ferreira	Sandia National Laboratories, USA
Ann Gentile	Sandia National Laboratories, USA
Cecile Germain	University Paris-Sud, France
Rinku Gupta	Argonne National Laboratory, USA
Paul Hargrove	Lawrence Berkeley National Laboratory, USA
Xubin He	Virginia Commonwealth University, USA
Daniel S. Katz	University of Chicago, USA
Larry Kaplan	Cray, USA
Thilo Kielmann	Vrije Universiteit Amsterdam, The Netherlands
Dieter Kranzlmüller	LMU/LRZ Munich, Germany

Chokchai (Box) Leangsuksun	Louisiana Tech University, USA
Xiaosong Ma	North Carolina State University, USA
Celso Mendes	University of Illinois at Urbana Champaign, USA
Christine Morin	INRIA Rennes, France
Thomas Naughton	Oak Ridge National Laboratory, USA
George Ostrouchov	Oak Ridge National Laboratory, USA
Mihaela Paun	Louisiana Tech University, USA
Alexander Reinefeld	Zuse Institute Berlin, Germany
Rolf Riesen	IBM Research, Ireland
Stephen L. Scott	Oak Ridge National Laboratory, USA
Gregory M. Thorson	SGI, USA
Geoffroy Vallee	Oak Ridge National Laboratory, USA
Sudharshan Vazhkudai	Oak Ridge National Laboratory, USA
Chao Wang	Oak Ridge National Laboratory, USA

5th Workshop on UnConventional High-Performance Computing (UCHPC 2012)

Program Chairs

Anders Hast	Uppsala University, Sweden
Josef Weidendorfer	Technische Universität München, Germany
Jan-Philipp Weiss	Karlsruhe Institute of Technology, Germany

Steering Committee

Lars Bengtsson	Chalmers University, Sweden
Anders Hast	Uppsala University, Sweden
Josef Weidendorfer	Technische Universität München, Germany
Jan-Philipp Weiss	KIT, Germany
Ren Wu	HP Labs, Palo Alto, USA

Program Committee

David A. Bader	Georgia Tech, USA
Michael Bader	Universität Stuttgart, Germany
Denis Barthou	Université de Bordeaux, France
Lars Bengtsson	Chalmers, Sweden
Giorgos Dimitrakopoulos	Democritus University of Thrace, Greece
Karl Fürlinger	LMU, Munich, Germany
Dominik Göddeke	TU Dortmund, Germany
Georg Hager	University Erlangen-Nuremberg, Germany
Anders Hast	Uppsala University, Sweden

Ben Juurlink	TU Berlin, Germany
Rainer Keller	HfT Stuttgart, Germany
Gaurav Khanna	University of Massachusetts Dartmouth, USA
Harald Köstler	University Erlangen-Nuremberg, Germany
Manfred Mücke	University of Vienna, Austria
Andy Nisbet	Manchester Metropolitan University, UK
Ioannis Papaefstathiou	Technical University of Crete, Greece
Franz-Josef Pfreundt	Fraunhofer ITWM, Germany
Bertil Schmidt	Johannes Gutenberg University Mainz, Germany
Dimitrios Soudris	National Technical University of Athens, Greece
Ioannis Soudris	Chalmers University of Technology, Sweden
Thomas Steinke	Zuse Institute, Berlin, Germany
Josef Weidendorfer	Technische Universität München, Germany
Jan-Philipp Weiss	KIT, Germany
Stephan Wong	Delft University of Technology, The Netherlands
Ren Wu	HP Labs Palo Alto, USA
Peter Zinterhof jun.	University of Salzburg, Austria
Yunquan Zhang	Chinese Academy of Sciences, Beijing, China

Additional Reviewers

Nicolas Schier	TU Berlin, Germany
----------------	--------------------

7th Workshop on Virtualization in High-Performance Cloud Computing (VHPC 2012)

Program Chairs

Michael Alexander	TU Wien, Vienna, Austria
Gianluigi Zanetti	CRS4, Italy
Anastassios Nanos	NTUA, Greece

Program Committee

Paolo Anedda	CRS4, Italy
Giovanni Busonera	CRS4, Italy
Brad Calder	Microsoft, USA
Roberto Canonico	University of Naples Federico II, Italy
Tommaso Cucinotta	Alcatel-Lucent Bell Labs, Ireland
Werner Fischer	Thomas-Krenn AG, Germany
William Gardner	University of Guelph, Canada
Marcus Hardt	Forschungszentrum Karlsruhe, Germany
Sverre Jarp	CERN, Switzerland

Shantenu Jha	Louisiana State University, USA
Xuxian Jiang	NC State, USA
Nectarios Koziris	National Technical University of Athens, Greece
Simone Leo	CRS4, Italy
Ignacio Llorente	Universidad Complutense de Madrid, Spain
Naoya Maruyama	Tokyo Institute of Technology, Japan
Jean-Marc Menaud	Ecole des Mines de Nantes, France
Dimitrios Nikolopoulos	Foundation for Research & Technology Hellas, Greece
Jose Renato Santos	HP Labs, USA
Walter Schwaiger	TU Wien, Austria
Yoshio Turner	HP Labs, USA
Kurt Tutschku	University of Vienna, Austria
Lizhe Wang	Indiana University, USA
Chao-Tung Yang	Tunghai University, Taiwan

Table of Contents

1st Workshop on Big Data Management in Clouds – BDMC2012

- 1st Workshop on Big Data Management in Clouds – BDMC2012 1
Alexandru Costan and Ciprian Dobre

- MRBS: Towards Dependability Benchmarking for Hadoop
MapReduce 3
Amit Sangroya, Damián Serrano, and Sara Bouchenak

- Caju: A Content Distribution System for Edge Networks 13
*Guthemberg Silvestre, Sébastien Monnet, Ruby Krishnaswamy, and
Pierre Sens*

- Data Security Perspectives in the Framework of Cloud Governance 24
Adrian Copie, Teodor-Florin Fortiș, and Victor Ion Munteanu

CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing – CGWS2012

- CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing –
CGWS2012 34
Frédéric Desprez, Domenico Talia, and Ramin Yahapour

- Evaluating Cloud Storage Services for Tightly-Coupled Applications 36
*Alexandra Carpen-Amarie, Kate Keahey, John Bresnahan, and
Gabriel Antoniu*

- Targeting Distributed Systems in FastFlow 47
*Marco Aldinucci, Sonia Campa, Marco Danelutto,
Peter Kilpatrick, and Massimo Torquati*

- Throughput Optimization for Pipeline Workflow Scheduling with Setup
Times 57
*Anne Benoit, Mathias Coqblin, Jean-Marc Nicod,
Laurent Philippe, and Veronika Rehn-Sonigo*

- Meteorological Simulations in the Cloud with the ASKALON
Environment 68
*Gabriela Andreea Morar, Felix Schüller, Simon Ostermann,
Radu Prodan, and Georg Mayr*

A Science-Gateway Workload Archive to Study Pilot Jobs, User Activity, Bag of Tasks, Task Sub-steps, and Workflow Executions	79
<i>Rafael Ferreira da Silva and Tristan Glatard</i>	
Energy Adaptive Mechanism for P2P File Sharing Protocols	89
<i>Mayank Raj, Krishna Kant, and Sajal K. Das</i>	
Tenth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar'2012)	
Tenth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar'2012)	100
<i>Rosa M. Badia</i>	
Unleashing CPU-GPU Acceleration for Control Theory Applications ...	102
<i>Peter Benner, Pablo Ezzatti, Enrique S. Quintana-Ortí, and Alfredo Remón</i>	
<i>clOpenCL - Supporting Distributed Heterogeneous Computing in HPC Clusters</i>	112
<i>Albano Alves, José Rufino, António Pina, and Luís Paulo Santos</i>	
Mastering Software Variant Explosion for GPU Accelerators	123
<i>Richard Membarth, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert</i>	
Exploring Heterogeneous Scheduling Using the Task-Centric Programming Model	133
<i>Artur Podobas, Mats Brorsson, and Vladimir Vlassov</i>	
Weighted Block-Asynchronous Iteration on GPU-Accelerated Systems	145
<i>Hartwig Anzt, Stanimire Tomov, Jack Dongarra, and Vincent Heuveline</i>	
An Optimized Parallel IDCT on Graphics Processing Units	155
<i>Biao Wang, Mauricio Alvarez-Mesa, Chi Ching Chi, and Ben Juurlink</i>	
Multi-level Parallelization of Advanced Video Coding on Hybrid CPU+GPU Platforms	165
<i>Svetislav Momcilovic, Nuno Roma, and Leonel Sousa</i>	
Multi-GPU Implementation of the NICAM Atmospheric Model	175
<i>Irina Demeshko, Naoya Maruyama, Hirofumi Tomita, and Satoshi Matsuoka</i>	

MPI vs. BitTorrent: Switching between Large-Message Broadcast Algorithms in the Presence of Bottleneck Links	185
<i>Kiril Dichev and Alexey Lastovetsky</i>	
MIP Model Scheduling for Multi-Clusters	196
<i>Héctor Blanco, Fernando Guirado, Josep Lluís Lérida, and V.M. Albornoz</i>	
Systematic Approach in Optimizing Numerical Memory-Bound Kernels on GPU	207
<i>Ahmad Abdelfattah, David Keyes, and Hatem Ltaief</i>	
HiBB 2012: 3 rd Workshop on High Performance Bioinformatics and Biomedicine	217
<i>Mario Cannataro</i>	
Using Clouds for Scalable Knowledge Discovery Applications	220
<i>Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio</i>	
P3S: Protein Structure Similarity Search	228
<i>Jakub Galgonek, Tomáš Skopal, and David Hoksza</i>	
On the Parallelization of the SProt Measure and the TM-Score Algorithm	238
<i>Jakub Galgonek, Martin Kruliš, and David Hoksza</i>	
Stochastic Simulation of the Coagulation Cascade: A Petri Net Based Approach	248
<i>Davide Castaldi, Daniele Maccagnola, Daniela Mari, and Francesco Archetti</i>	
Processing the Biomedical Data on the Grid Using the UNICORE Workflow System	263
<i>Marcelina Borcz, Rafał Kluszczyński, Katarzyna Skonieczna, Tomasz Grzybowski, and Piotr Bała</i>	
Multicore and Cloud-Based Solutions for Genomic Variant Analysis	273
<i>Cristina Y. González, Marta Bleda, Francisco Salavert, Rubén Sánchez, Joaquín Dopazo, and Ignacio Medina</i>	
A Novel Implementation of Double Precision and Real Valued ICA Algorithm for Bioinformatics Applications on GPUs	285
<i>Amin Foshati and Farshad Khunjush</i>	

PROGENIA: An Approach for Grid Interoperability at Workflow Level	295
<i>Maria Mirto, Marco Passante, and Giovanni Aloisio</i>	
OMHI 2012: First International Workshop on On-chip Memory Hierarchies and Interconnects: Organization, Management and Implementation	
OMHI 2012: First International Workshop on On-chip Memory Hierarchies and Interconnects: Organization, Management and Implementation	305
<i>Julio Sahuquillo, María E. Gómez, and Salvador Petit</i>	
Allocating Irregular Partitions in Mesh-Based On-Chip Networks	307
<i>Samuel Rodrigo, Frank Olaf Sem-Jacobsen, and Tor Skeie</i>	
Detecting Sharing Patterns in Industrial Parallel Applications for Embedded Heterogeneous Multicore Systems	317
<i>Albert Esteve, María Soler, María Engracia Gómez, Antonio Robles, and José Flích</i>	
Addressing Link Degradation in NoC-Based ULSI Designs	327
<i>Carles Hernández, Federico Silla, and José Duato</i>	
Performance and Energy Efficiency Analysis of Data Reuse Transformation Methodology on Multicore Processor	337
<i>Abdullah Al Hasib, Per Gunnar Kjeldsberg, and Lasse Natvig</i>	
Effects of Process Variation on the Access Time in SRAM Cells	347
<i>Vicent Lorente and Julio Sahuquillo</i>	
Task Scheduling on Manycore Processors with Home Caches	357
<i>Ananya Muddukrishna, Artur Podobas, Mats Brorsson, and Vladimir Vlassov</i>	
ParaPhrase Workshop 2012	
ParaPhrase Workshop 2012	368
<i>M. Danelutto, K. Hammond, and H. Gonzalez-Velez</i>	
Using the SkelCL Library for High-Level GPU Programming of 2D Applications	370
<i>Michel Steuwer, Sergei Gorlatch, Matthias Buß, and Stefan Breuer</i>	
Structured Data Access Annotations for Massively Parallel Computations	381
<i>Marco Aldinucci, Sonia Campa, Peter Kilpatrick, and Massimo Torquati</i>	

PROPER 2012: Fifth Workshop on Productivity and Performance – Tools for HPC Application Development

PROPER 2012: Fifth Workshop on Productivity and Performance – Tools for HPC Application Development	391
<i>Bettina Krammer</i>	
Performance Engineering: From Numbers to Insight	393
<i>Georg Hager</i>	
Runtime Function Instrumentation with EZTrace	395
<i>Charles Aulagnon, Damien Martin-Guillerez, François Rué, and François Trahay</i>	
Compiler Help for Binary Manipulation Tools	404
<i>Tugrul Ince and Jeffrey K. Hollingsworth</i>	
On the Instrumentation of OpenMP and OmpSs Tasking Constructs ...	414
<i>Harald Servat, Xavier Teruel, Germán Llort, Alejandro Duran, Judit Giménez, Xavier Martorell, Eduard Ayguadé, and Jesús Labarta</i>	
Strategies for Real-Time Event Reduction	429
<i>Michael Wagner and Wolfgang E. Nagel</i>	
A Scalable InfiniBand Network Topology-Aware Performance Analysis Tool for MPI	439
<i>Hari Subramoni, Jerome Vienne, and Dhabaleswar K. (DK) Panda</i>	
Performance Patterns and Hardware Metrics on Modern Multicore Processors: Best Practices for Performance Engineering	451
<i>Jan Treibig, Georg Hager, and Gerhard Wellein</i>	
Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids	
Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids	461
<i>Stephen L. Scott and Chokchai (Box) Leangsuksun</i>	
High Performance Reliable File Transfers Using Automatic Many-to-Many Parallelization	463
<i>Paul Z. Kolano</i>	

A Reliability Model for Cloud Computing for High Performance Computing Applications	474
<i>Thanadech Thanakornworakij, Raja F. Nassar, Chokchai Leangsuksun, and Mihaela Păun</i>	
The Viability of Using Compression to Decrease Message Log Sizes	484
<i>Kurt B. Ferreira, Rolf Riesen, Dorian Arnold, Dewan Ibtesham, and Ron Brightwell</i>	
Resiliency in Exascale Systems and Computations Using Chaotic-Identity Maps	494
<i>Nageswara S.V. Rao</i>	
Programming Model Extensions for Resilience in Extreme Scale Computing	496
<i>Saurabh Hukerikar, Pedro C. Diniz, and Robert F. Lucas</i>	
User Level Failure Mitigation in MPI	499
<i>Wesley Bland</i>	

UCHPC 2012: Fifth Workshop on UnConventional High Performance Computing

UCHPC 2012: Fifth Workshop on UnConventional High Performance Computing	505
<i>Anders Hast, Josef Weidendorfer, and Jan-Philipp Weiss</i>	
A Methodology for Efficient Use of OpenCL, ESL and FPGAs in Multi-core Architectures	507
<i>Alexandros Bartzas and George Economakos</i>	
Efficient Design Space Exploration of GPGPU Architectures	518
<i>Ali Jooya, Amirali Baniasadi, and Nikitas J. Dimopoulos</i>	
Spin Glass Simulations on the Janus Architecture: A Desperate Quest for Strong Scaling	528
<i>M. Baity-Jesi, R.A. Baños, A. Cruz, L.A. Fernandez, J.M. Gil-Narvion, A. Gordillo-Guerrero, M. Guidetti, D. Iñiguez, A. Maiorano, F. Mantovani, E. Marinari, V. Martin-Mayor, J. Monforte-Garcia, A. Muñoz-Sudupe, D. Navarro, G. Parisi, S. Perez-Gavirio, M. Pivanti, F. Ricci-Tersenghi, J. Ruiz-Lorenzo, S.F. Schifano, B. Seoane, A. Tarancon, P. Tellez, R. Tripiccione, and D. Yllanes</i>	

7th Workshop on Virtualization in High-Performance Cloud Computing – VHPC2012

7th Workshop on Virtualization in High-Performance Cloud Computing – VHPC2012.....	538
<i>Michael Alexander, Gianluigi Zanetti, and Anastassios Nanos</i>	
Pre-Copy and Post-Copy VM Live Migration for Memory Intensive Applications.....	539
<i>Aidan Shribman and Benoit Hudzia</i>	
Xen2MX: Towards High-Performance Communication in the Cloud.....	548
<i>Anastassios Nanos and Nectarios Koziris</i>	
Themis: Energy Efficient Management of Workloads in Virtualized Data Centers	557
<i>Gaurav Dhiman, Vasileios Kontorinis, Raid Ayoub, Liuyi Zhang, Chris Sadler, Dean Tullsen, and Tajana Simunic Rosing</i>	
Runtime Virtual Machine Recontextualization for Clouds	567
<i>Django Armstrong, Daniel Espling, Johan Tordsson, Karim Djemame, and Erik Elmroth</i>	
GaaS: Customized Grids in the Clouds	577
<i>G.B. Barone, R. Bifulco, V. Boccia, D. Bottalico, R. Canonico, and L. Carracciuolo</i>	
Author Index	587

1st Workshop on Big Data Management in Clouds – BDMC2012

Alexandru Costan¹ and Ciprian Dobre²

¹ Inria Rennes - Bretagne Atlantique, France

² University Politehnica of Bucharest, Romania

As data volumes increase at exponential speed in more and more application fields of science, the challenges posed by handling Big Data in the Exabyte era gain an increasing importance. High-energy physics, statistics, climate modeling, cosmology, genetics or bio-informatics are just a few examples of fields where it becomes crucial to efficiently manipulate Big Data, which are typically shared at large scale. Rapidly storing this data, protecting it from loss and analyzing it to understand the results are significant challenges, made more difficult by decades of improvements in computation capabilities that have been unmatched in storage. For many applications, the overall performance and scalability becomes clearly driven by the performance of the data handling subsystem. As we anticipate Exascale systems in 2020, there is a growing consensus in the scientific community that revolutionary new approaches are needed in computational science data management. These new trends lead us to rethink the traditional file-based data management abstraction for large-scale applications. Moreover, for obvious cost-related reasons, new architectures are clearly needed as well as alternate infrastructures to supercomputers., like hybrid or HPC clouds.

The Workshop on Big Data Management in Clouds was created to provide a platform for the dissemination of recent research efforts that explicitly aim at addressing these challenges. It supports the presentation of advanced solutions for the efficient management of Big Data in the context of Cloud computing. The workshop aims to provide a venue for researchers to present and discuss results on all aspects of data management in Clouds, new development and deployment efforts in running data-intensive computing workloads. In particular, we are interested in how the use of Cloud-based technologies can meet the data intensive scientific challenges of HPC applications that are not well served by the current supercomputers or grids, and are being ported to Cloud platforms. The goal of the workshop is to support the assessment of the current state, introduce future directions, and present architectures and services for future Clouds supporting data intensive computing. The proposal of this workshop is a result of the ongoing collaboration existing between Inria and the Politehnica University of Bucharest in the framework of the DataCloud@work Associate Team (<http://www.irisa.fr/kerdata/>) and of the FP7 ERRIC project (<http://erric.eu/>). It has also links with the ANR MapReduce (<http://mapreduce.inria.fr>) and FP7 MCITN SCALUS (<http://www.scalus.eu/>) projects.

The first edition of the workshop was held on August 27, 2012 jointly with the CoreGRID/ERCIM Workshop and gathered around 40 researchers from academia and industry. We received a total of six papers, out of which three were selected for presentation after one keynote talk. The keynote talk was given by Marco Aldinucci, Assistant Professor at the Computer Science Department of University of Torino and highlighted several new techniques and tools for parallel computing on online data streams in systems biology and epidemiology. The talk focused on extracting relevant knowledge from Big Data: it described streaming and online data filtering in parallel computing as a way to both ameliorate the I/O bottleneck and to raise the abstraction level in software construction, enhancing performance portability and time-to-solution.

The first paper presented a benchmark suite for evaluating the dependability of MapReduce systems through a wide range of execution scenarios such as data-intensive vs. compute-intensive applications, or batch applications vs. online interactive applications. The system allows to inject various types of faults at different rates and produces extensive reliability, availability and performance statistics. It was illustrated with an online Hadoop MapReduce cluster as a software framework to help researchers and practitioners to better analyze and evaluate the fault-tolerance of their systems. The second paper presented a content distribution system for edge networks, relying on an architecture that gathers several storage domains composed of small-sized datacenters and edge devices. The system provides the ability to manage storage and network resources from both consumer and operators in a collaborative manner. The evaluations showed that non-collaborative caching consistently outperforms the fixed replication scheme, proving the importance of adapting the replication degree to data popularity. The last paper focused on determining the requirements that must be met by the various databases in a complex datastore used in the cloud, emphasizing the threats and security risks that the individual database entities must face.

We wish to thank all the authors, the keynote speaker, the PC members and the organizers of EuroPar 2012 for their contribution to the success of this edition of the Workshop.

MRBS: Towards Dependability Benchmarking for Hadoop MapReduce

Amit Sangroya, Damián Serrano, and Sara Bouchenak

University of Grenoble - LIG - INRIA, Grenoble, France
`{Amit.Sangroya,Damian.Serrano,Sara.Bouchenak}@inria.fr`

Abstract. MapReduce is a popular programming model for distributed data processing. Extensive research has been conducted on the reliability of MapReduce, ranging from adaptive and on-demand fault-tolerance to new fault-tolerance models. However, realistic benchmarks are still missing to analyze and compare the effectiveness of these proposals. To date, most MapReduce fault-tolerance solutions have been evaluated using microbenchmarks in an ad-hoc and overly simplified setting, which may not be representative of real-world applications. This paper presents MRBS, a comprehensive benchmark suite for evaluating the dependability of MapReduce systems. MRBS includes five benchmarks covering several application domains and a wide range of execution scenarios such as data-intensive vs. compute-intensive applications, or batch applications vs. online interactive applications. MRBS allows to inject various types of faults at different rates and produces extensive reliability, availability and performance statistics. The paper illustrates the use of MRBS with Hadoop clusters.

Keywords: Benchmark, Dependability, MapReduce, Hadoop.

1 Introduction

MapReduce has become a popular programming model and runtime environment for developing and executing distributed data-intensive and compute-intensive applications [1]. It offers developers a means to transparently handle data partitioning, replication, task scheduling and fault-tolerance on a cluster of commodity computers. Hadoop [2], one of the most popular MapReduce frameworks, provides key fault-tolerance features.

There has been a large amount of work towards improving fault-tolerance solutions in MapReduce. Several efforts have explored on-demand fault-tolerance [3], replication and partitioning policies [4], [5], adaptive fault-tolerance [6], [7], and extending MapReduce with other fault-tolerance models [8], [9]. However, there has been very little in the way of empirical evaluation of MapReduce dependability. Evaluations have often been conducted in an ad-hoc manner, such as turning off a node in the MapReduce cluster or killing a task process. Recent tools, like Hadoop fault injection framework offer the ability to emulate non-deterministic exceptions in the distributed filesystem. Although they provide a

means to program unit tests for HDFS, such low-level tools are meant to be used by developers who are familiar with the internals of HDFS, and are unlikely to be used by end-users of MapReduce systems. MapReduce fault injection must therefore be generalized and automated for higher-level and easier use. Not only it is necessary to automate the injection of faults, but also the definition and generation of MapReduce faultloads. A faultload describes *what* fault to inject (e.g. a node crash), *where* to inject it (e.g. which node of the MapReduce cluster), and *when* to inject it (e.g. five minutes after the application started).

This paper presents MRBS (MapReduce Benchmark Suite), for evaluating the dependability of MapReduce systems. MRBS enables automatic faultload generation and injection in MapReduce. This covers different fault types, injected at different rates, which will provide a means to analyze the effectiveness of fault-tolerance in a variety of scenarios. Three different aspects of load are considered: dataload, workload and faultload. MRBS allows to quantify dependability levels provided by MapReduce fault-tolerance systems, through an empiric evaluation of the availability and reliability of such systems, in addition to performance and cost metrics. Moreover, MRBS covers five application domains: recommendation systems, business intelligence, bioinformatics, text processing, and data mining. It supports a variety of workload and dataload characteristics, ranging from compute-oriented to data-oriented applications, batch applications to online interactive applications. Indeed, while MapReduce frameworks were originally limited to offline batch applications, recent works are exploring the extension of MapReduce beyond batch processing [10].

MRBS shows that when running the Bioinformatics workload and injecting a faultload that consists of a hundred map software faults and three node faults, Hadoop MapReduce handles these failures with high reliability (94% of successful requests) and high availability (96% of the time). We wish to make dependability benchmarking easy to adopt by end-users of MapReduce and developers of MapReduce fault-tolerance systems. MRBS allows automatic deployment of experiments on private or public clouds being independent of any particular infrastructure. MRBS is available as a software framework to help researchers and practitioners to better analyze and evaluate the dependability and performance of MapReduce systems. It can be downloaded from <http://sardes.inrialpes.fr/research/mrbs>.

2 Background on Hadoop Fault Tolerance

MapReduce is a programming model and a software framework introduced by Google in 2004 to support distributed computing and large data processing on clusters of commodity machines [1]. MapReduce supports a wide range of applications such as image analytics, next-generation sequencing, recommendation systems, search engines, social networks, business intelligence, and log analysis.

There are many implementations of MapReduce among which the popular open-source Hadoop framework, which is also available in public clouds such as Amazon EC2 or Open Cirrus. A Hadoop cluster consists of a *master node*

and *slave nodes*. Users (i.e. clients) of a Hadoop cluster submit MapReduce jobs to the master node which hosts the *JobTracker* daemon that is responsible of scheduling the jobs. Each slave node hosts a *TaskTracker* daemon that periodically communicates with the master node to indicate whether the slave is ready to run new tasks. If it is, the master schedules appropriate tasks on the slave.

Hadoop framework also provides a distributed filesystem (HDFS) that stores data across cluster nodes. HDFS architecture consists of a *NameNode* and *DataNodes*. The *NameNode* daemon runs on the master node and is responsible of managing the filesystem namespace and regulating access to files. A *DataNode* daemon runs on a slave node and is responsible of managing storage attached to that node. HDFS is thus a means to store input, intermediate and output data of Hadoop MapReduce jobs. Furthermore, for fault tolerance purposes, HDFS replicates data on different nodes.

One of the major features of Hadoop MapReduce is its ability to tolerate failures of different types, as described in the following.

Node Crash: In case of a slave node failure, the JobTracker on the master node stops receiving heartbeats from the TaskTracker on the slave for an interval of time. When it notices the failure of a slave node, the master removes the node from its pool and reschedules ongoing tasks on other nodes.

Task Process Crash: A task may also fail because a map or reduce task process suddenly crashes, e.g., due to a transient bug in the underlying (virtual) machine. Here again, the parent TaskTracker notices that a task process has exited and notifies the JobTracker for possible task retries.

Task Software Fault: A task may fail due to errors and runtime exceptions in *map* or *reduce* functions written by the programmer. When a TaskTracker on a slave node notices that a task it hosts has failed, it notifies the JobTracker which reschedules another execution of the task, up to a maximum number of retries.

Hanging Tasks: A map or reduce task is marked as failed if it stops sending progress updates to its parent TaskTracker for a period of time (indicated by *mapred.task.timeout* Hadoop property). If that occurs, the task process is killed, and the JobTracker is notified for possible task retries.

3 Dependability Benchmarking for Hadoop MapReduce

To use MRBS, three main steps are needed: (i) build a faultload (i.e. fault scenario) to describe the set of faults to be injected, (ii) conduct fault injection experiments based on the faultload, and (iii) collect statistics about dependability levels of the MapReduce system under test. This is presented in Figure 1.

The evaluator of the dependability of a MapReduce system chooses an application from MRBS' set of benchmarks, depending on the desired application domain and whether he/she targets compute-oriented or data-oriented applications. MRBS injects (possibly default) workload and dataload in the system under test. MRBS also allows the evaluator to choose specific dataload and workload, to stress the scalability of the MapReduce system (see Sections 3.3 and 3.4 for more details).

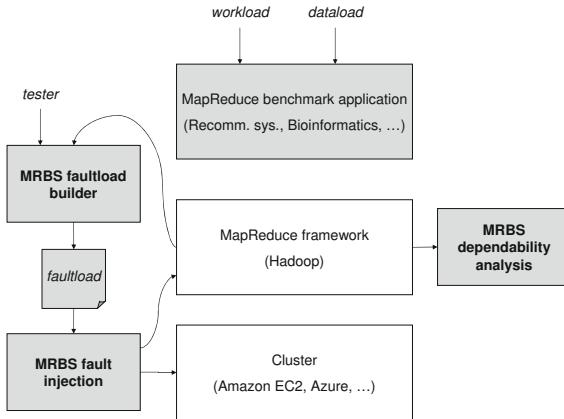


Fig. 1. Overview of MRBS dependability benchmarking

3.1 Faultload Builder

A faultload in MRBS is described in a file, either by extension, or by intention. In the former case, each line of the faultload file consists of the following elements: the time at which a fault occurs , the type of fault that occurs and, optionally, where the fault occurs. A fault belongs to one of the fault types handled by Hadoop MapReduce¹. Another way to define a more concise faultload is to describe it by intention. Here, each line of the faultload file consists of: a fault type, and the mean time between failures (MTBF) of that type. Thus, testers can explicitly build synthetic faultloads representing various fault scenarios.

A faultload description may also be automatically obtained, either randomly or based on previous application runs' traces. The random faultload builder produces a faultload description where, with each fault type, is associated a random MTBF between 0 and the length of the experiment. Similarly, the random faultload builder may produce a faultload by extension, where it generates the i^{th} line of the faultload file as follows: $< \text{time_stamp}_i, \text{fault_type}_i, \text{fault_location}_i >$, with time_stamp_i being a random value between time_stamp_{i-1} (or 0 if $i = 1$) and the length of the experiment, fault_type_i and fault_location_i random values in the set of possible values. A faultload description may also be automatically generated based on traces of previous runs of MapReduce applications.

MRBS faultload builder is relatively portable: its two first variants – explicit faultload builder and random faultload builder – are general enough and do not rely on any specific platform. The trace-based faultload builder is independent from the internals of the MapReduce framework, and produces a faultload description based on the structure of the MapReduce framework's logs; it currently works on Hadoop MapReduce framework.

¹ Other types of faults, such as network disconnection, may be emulated by MRBS, although we do not detail them in this paper.

3.2 Fault Injection

The output of the MRBS faultload builder is passed to the MRBS fault injector. The MRBS fault injector divides the input faultload into subsets of faultloads: one crash faultload groups all crash faults that will occur in all nodes of the MapReduce cluster (i.e. node crash, task process crash), and one per-node faultload groups all occurrences of other types of faults that will occur in one node (i.e. task software faults, hanging tasks).

The MRBS fault injector runs a daemon that is responsible of injecting the crash faultload. In the following, we present how the daemon injects these faults, in case of a faultload described by extension, although this can be easily generalized to a faultload described by intention. Thus, for the i^{th} fault in the crash faultload, the daemon waits until $time_stamp_i$ is reached, then calls the fault injector of $fault_type_i$ (see below), on the MapReduce cluster node corresponding to $fault_location_i$. This fault injector is called as many times as there are occurrences of the same fault at the same time. The fault injection daemon repeats these operations for the following crash faults, until the end of the faultload file is reached or the experiment finishes.

The MRBS fault injector handles the per-node faultloads differently. A per-node faultload includes faults that occur inside tasks. MRBS intercepts task creation to check whether a fault must be injected in that task, in which case the fault injector corresponding to the fault type is called (see below). MRBS does not require the modification of the source code of the MapReduce framework. Instead, it synthesizes a new version of the MapReduce framework library using aspect-oriented techniques. The synthetic MapReduce library has the same API as the original one, but underneath this new library includes task creation interceptors that encode the fault injection logic.

Node Crash Injection: A node crash is simply implemented by shutting down a node. This fault injector uses the API of the underlying cloud infrastructure to implement such a fault. For example, in case of a public cloud such as Amazon EC2, a node crash consists in a premature termination of an Amazon EC2 instance. However, if a tester wants to conduct multiple runs of the same dependability experiment, and if faults are implemented by shutting down machines, new machines must be acquired from the cloud at the beginning of each run, which may induce a delay. For efficiency purposes, we propose an implementation of MapReduce node fault which kills all MapReduce daemons running on that node. Specifically, in the case of Hadoop these include the *TaskTracker* and *DataNode* daemons running in a slave node². The timeout to detect a MapReduce node failure is set to 30 seconds, a value set in *mapred.task.tracker.expiry.interval* Hadoop property.

Task Process Crash Injection: This type of fault is implemented by killing the process running a task on a MapReduce node.

² A node crash is not injected to the MapReduce master node since this node is not fault-tolerant.

Task Software Fault Injection: A task software fault is implemented as a runtime exception thrown by a map task or a reduce task. This fault injector is called by the interceptors injected into the MapReduce framework library by MRBS.

Provoking Hanging Tasks: A task is marked as hanging if it stops sending progress updates for a period of time. This type of fault is injected into a map task or a reduce task through the interceptors that make the task sleep a longer time than the maximum period of time for sending progress updates (*mapred.task.timeout* Hadoop property).

The MRBS faultload injector is relatively portable: it is independent from the internals of the MapReduce framework and the per-node faultload injectors are automatically integrated within the framework based upon its API. The current version of the MRBS faultload injector works for Hadoop MapReduce; porting to new platforms is straightforward.

3.3 Benchmark Suite

Conceptually, a benchmark in MRBS implements a service that provides different types of operations, which are requested by clients. The benchmark service is implemented as a set of MapReduce programs running on a cluster, and clients are implemented as external entities that remotely request the service. Depending on the complexity of a client request, the request may consist of one or multiple successive MapReduce jobs. A benchmark has two execution modes: interactive mode or batch mode. In interactive mode, concurrent clients share the MapReduce cluster at the same time (i.e. have their requests executed concurrently). On the other hand, requests from different clients are executed in FIFO order (one after another) in batch mode.

A benchmark run has three successive phases: a warm-up phase, a run-time phase, and a slow-down phase, which length may be chosen by the end-user of the benchmark. The end-user may also choose the number of times a benchmark is run, to produce average statistics. MRBS benchmark suite consists of five benchmarks covering various application domains such as recommendation systems, business intelligence, bioinformatics, text processing, and data mining. The user can choose the actual benchmark.

Recommendation System: Recommendation systems are widely used in e-commerce sites. MRBS implements an online movie recommender system. It builds upon a set of movies, a set of users, and a set of ratings and reviews users give for movies to indicate whether and how much they liked or disliked the movies.

Business Intelligence: The Business Intelligence benchmark represents a decision support system for a wholesale supplier. It implements business-oriented queries that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. It uses Apache Hive on top of Hadoop, a data warehouse that facilitates ad-hoc queries using a SQL-like language called HiveQL.

Bioinformatics: The benchmark includes a MapReduce-based implementation of DNA sequencing. The data used in the benchmark are publicly available genomes. Currently, the benchmark allows to analyze several genomes of organisms such as the pathogenic organisms *Salmonella typhi*, *Rhodococcus equi*, and *Streptococcus suis*.

Text Processing: Text processing is a classical application of MapReduce. MRBS provides a MapReduce text processing-oriented benchmark, with three types of operations allowing clients to search words or word patterns in text documents, to know how often words occur in text documents, or to sort the contents of documents. The benchmark uses synthetic input data that consist of randomly generated text files of different sizes.

Data Mining: This benchmark provides two types of data mining operations: clustering and classification. MRBS benchmark considers the case of classifying newsgroup documents into categories. Furthermore, the benchmark provides canopy clustering operations.

3.4 Using MRBS

MRBS comes with a configuration file that involves several parameters among which the following: the actual benchmark to use, the length of the benchmark warm-up phase, runtime phase, and slow-down phase, the size of the benchmark input data set, the size of the MapReduce cluster, the cloud infrastructure that will host the cluster, in addition to workload and faultload characteristics described in the previous section. To keep the use of MRBS simple, these parameters have default values that may be adjusted by MRBS user.

MRBS produces various runtime statistics related to performance and dependability. These include client request response time, throughput, financial cost, failed client requests vs. successful requests, availability, and reliability. MRBS also provides low-level MapReduce statistics related to the number, length and status (i.e. success or failure) of MapReduce jobs; tasks; the size of data read from or written to the distributed file system, etc. These low-level statistics are built using Hadoop counters. Optionally, MRBS can generate charts plotting continuous-time results. More details on MRBS can be found in [11].

4 Evaluation

4.1 Experimental Setup

We conducted several experiments with MRBS on Hadoop clusters running in Amazon EC2 and Grid'5000, but due to space limitation we only present one case study in this paper. The experiments presented in this section were conducted in a cluster running in Grid'5000, a French geographically distributed infrastructure used to study large-scale parallel and distributed systems. The hardware configuration consists of 4-core 2-CPU, 2.5 GHz Intel Xeon E5420 QC

CPU, 8 GB memory, 160 GB SATA storage (per node) and 1 GB Ethernet network. The cluster consists of one node hosting MRBS and emulating concurrent clients, and a set of nodes hosting the MapReduce cluster.

Financial cost is \$0.34 per instance-hour. In the following, each experiment is run three times to report average and standard deviation results. The operating system of the nodes is Debian Linux 6 with kernel v2.6.32. The MapReduce framework is Apache Hadoop v0.20.2, and Hive v0.7, on Java 6.

4.2 Experimental Results

In this section, we illustrate the use of MRBS to evaluate the fault-tolerance of Hadoop MapReduce. Here, a ten-node Hadoop cluster runs the Bioinformatics benchmark, used by 20 concurrent clients. The experiment is conducted during a run-time phase of 60 minutes, after a warm-up phase of 15 minutes. We consider a synthetic faultload that consists of software faults and hardware faults as follows: first, 100 map task software faults are injected 5 minutes after the beginning of the run-time phase, and then, 3 node crashes are injected 25 minutes later.

Table 1. Reliability, availability, and cost

Reliability	Availability	Cost (dollars/request)
94%	96%	0.008 (+14%)

Although the injected faultload is aggressive, the Hadoop cluster remains available 96% of the time, and is able to successfully handle 94% of client requests (see Table 1). This has an impact on the request cost which is 14% higher than the cost obtained with the baseline (non-faulty) system.

To better explain the behavior of the MapReduce cluster, we will analyze MapReduce statistics, as presented in Figures 2(a) and 2(b). Figure 2(a) presents successful MapReduce jobs and failed MapReduce jobs over time. Note the logarithmic scale of the right side y-axis. When software faults occur, few jobs actually fail. On the contrary, node crashes are more damaging and induce a higher number of job failures, with a drop of the throughput of successful jobs from 16 jobs/minute before node failures to 5 jobs/minute after node failures.

Figure 2(b) shows the number of successful MapReduce tasks and the number of failed tasks over time, differentiating between tasks that fail because they are unable to access data from the underlying filesystem (i.e. I/O failures in the Figure), and tasks that fail because of runtime errors in all task retries (i.e. task failures in the Figure). We notice that software faults induce task failures that appear at the time the software faults occur, whereas node crashes induce I/O failures that last fifteen minutes after the occurrence of node faults. Actually, when some cluster nodes fail, Hadoop must reconstruct the state of the filesystem, by re-replicating the data blocks that were on the failed nodes from replicas in other nodes of the cluster. This explains the delay during which I/O failures are observed.

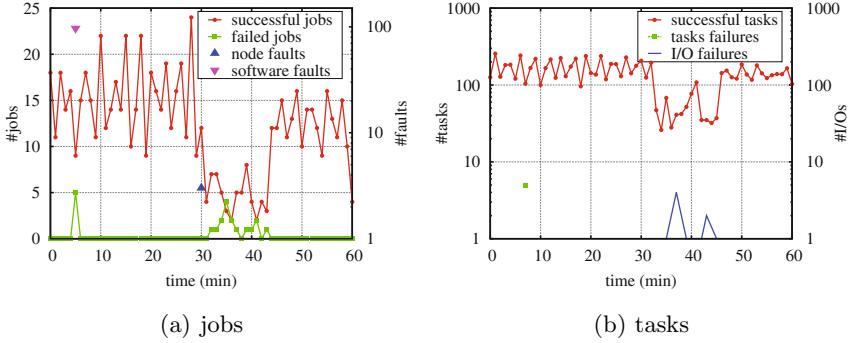


Fig. 2. Successful vs. failed MapReduce jobs and tasks

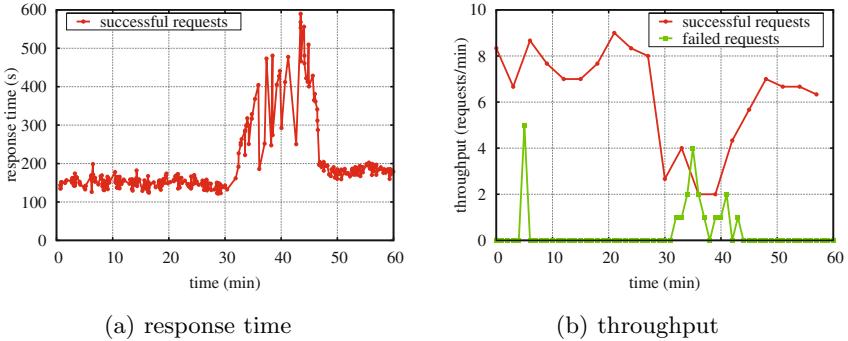


Fig. 3. Client request response time and throughput

Figure 3(a) shows the response time of successful client requests. With software faults, there is no noticeable impact on response times. Conversely, response time sharply increases when there are node faults, and while Hadoop is rebuilding missing data replicas. Similarly, Figure 3(b) presents the impact of failures on client request throughput. Interestingly, when the Hadoop cluster loses 3 nodes, it is able to fail-over, however, at the expense of a higher response time (+30%) and a lower throughput (-12%).

5 Conclusions and Perspectives

To evaluate the dependability of MapReduce systems, MRBS allows to characterize a faultload, generate it, and inject it in an online Hadoop MapReduce cluster. MRBS performs an empirical evaluation of the availability and reliability of such systems, to quantify their dependability levels. MRBS is available as a software framework to help researchers and practitioners to better analyze and evaluate the fault-tolerance of MapReduce systems. Important perspectives

of this work is the addition of security evaluation that supports data security attacks injection scenarios.

Acknowledgments. This work was partly supported by the Agence Nationale de la Recherche, the French National Agency for Research, under the MyCloud ANR project, and the University of Grenoble. Part of the experiments were conducted on the Grid'5000 experimental testbed, developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies.

References

1. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI (2004)
2. Apache Hadoop, <http://hadoop.apache.org>
3. Fadika, Z., Govindaraju, M.: LEMO-MR: Low Overhead and Elastic MapReduce Implementation Optimized for Memory and CPU-Intensive Applications. In: IEEE CloudCom (2010)
4. Ananthanarayanan, G., Agarwal, S., Kandula, S., Greenberg, A., Stoica, I., Harlan, D., Harris, E.: Scarlett: Coping with Skewed Content Popularity in MapReduce Clusters. In: European Conf. on Computer Systems (EuroSys) (2011)
5. Eltabakh, M., Tian, Y., Ozcan, F., Gemulla, R., Krettek, A., McPherson, J.: Co-Hadoop: Flexible Data Placement and Its Exploitation in Hadoop. In: VLDB (2011)
6. Jin, H., Yang, X., Sun, X.H., Raicu, I.: ADAPT: Availability-Aware MapReduce Data Placement in Non-Dedicated Distributed Computing Environment. In: ICDCS (2012)
7. Lin, H., Ma, X., Archuleta, J., Feng, W.C., Gardner, M., Zhang, Z.: MOON: MapReduce On Opportunistic eNvironments. In: HPDC (2010)
8. Bessani, A.N., Cogo, V.V., Correia, M., Costa, P., Pasin, M., Silva, F., Arantes, L., Marin, O., Sens, P., Sopena, J.: Making Hadoop MapReduce Byzantine Fault-Tolerant. In: DSN, Fast abstract (2010)
9. Ko, S.Y., Hoque, I., Cho, B., Gupta, I.: Making Cloud Intermediate Data Fault-Tolerant. In: ACM Symp. on Cloud Computing (SoCC) (2010)
10. Liu, H., Orban, D.: Cloud MapReduce: A MapReduce Implementation on Top of a Cloud Operating System. In: CCGRID (2011)
11. Sangroya, A., Serrano, D., Bouchenak, S.: MRBS: A Comprehensive MapReduce Benchmark Suite. Research Report RR-LIG-024, LIG, Grenoble, France (February 2012)

Caju: A Content Distribution System for Edge Networks

Guthemberg Silvestre^{1,2}, Sébastien Monnet¹, Ruby Krishnaswamy²,
and Pierre Sens¹

¹ LIP6/UPMC/CNRS/INRIA

4 place Jussieu, 75005 Paris, France

² Orange Labs,

38-40, rue du Général Leclerc, 92130 Issy-les-Moulineaux, France

Abstract. More and more, users store their data in the cloud. While the content is then retrieved, the retrieval has to respect quality of service (QoS) constraints. In order to reduce transfer latency, data is replicated. The idea is make data close to users and to take advantage of providers home storage. However to minimize the cost of their platform, cloud providers need to limit the amount of storage usage. This is still more crucial for big contents.

This problem is hard, the distribution of the popularity among the stored pieces of data is highly non-uniform: several pieces of data will never be accessed while others may be retrieved thousands of times. Thus, the trade-off between storage usage and QoS of data retrieval has to take into account the data popularity.

This paper presents our architecture gathering several storage domains composed of small-sized datacenters and edge devices; and it shows the importance of adapting the replication degree to data popularity.

Our simulations, using realistic workloads, show that a simple cache mechanism provides a eight-fold decrease in the number of SLA violations, requires up to 10 times less of storage capacity for replicas, and reduces aggregate bandwidth and number of flows by half.

1 Introduction

Content distribution over the internet has increased dramatically in the recent years. A recent study published by Cisco System, Inc [2] revealed that the global internet video traffic has surpassed peer-to-peer traffic since 2010, becoming the largest internet traffic type. Cisco Systems also forecasts that internet video traffic will reach 62% of the consumer internet traffic by 2015. The vast majority of this traffic consists of big popular content transport, including high-quality videos.

Nowadays, a large amount of data is stored “in the network”. This allows users to ease data sharing and retrieval, anywhere in the world. In the cloud, customers and providers come with storage service guarantees, such as QoS metrics, drawn up in Service Level Agreement (SLA) contracts. The provider is

therefore responsible to ensure data durability and availability. To enforce SLAs, providers rely on content replication. Yet, they need to do this carefully, it can have a huge impact on storage and bandwidth consumptions, even more for big contents. As data popularity is highly non-uniform, it is important to avoid replicating unpopular data, that will never be accessed, and also to ensure enough number of replicas for a popular content which may be retrieved concurrently by hundreds of users.

This work introduces and evaluates Caju, a content distribution system for edge networks. We analyse the performance of Caju as infrastructure for offering elastic storage cloud to users. We assume that cloud users may be eager to watch high-quality videos on-demand, on which strict SLA contracts have to be enforced. We study the impact of adding strict data transfer rates, as the main QoS metric, for SLA contracts in the cloud. We evaluate through simulations two replication schemes with synthetic traces that fairly reproduces big data requests, including popularity growth.

This work makes two main contributions:

- We describe the design, model, and implementation of Caju, a content distribution system for edge networks, that provides simple replication mechanisms, and allow us to manage edge resources properly.
- We evaluate the impact of big popular content on replication schemes in order to provide elastic storage to cloud users with regard to strict SLA contracts.

The rest of this work is organized as follows. Section 2 covers some background of the today’s content distribution systems and related work. Section 3 presents our approach to tackle elastic storage provision on the edge of the network, and provides an in-depth description of Caju, our system for CDNs at edge-networks. Section 4 analyses and explains our evaluation scenario and performance results. Finally, Section 5 shows future work and concludes the paper.

2 Background and State of the art

We first describe the current scenario of content distribution networks and the role of edge networks in the content distribution. Then, we focus on replication systems used by cloud and P2P storage systems.

Content Distribution Networks and Edge Networks: Content distributions networks (CDN) are distributed systems that maintain content servers in many different locations in order to improve content dissemination efficiency, enhance QoS metrics for end-users, and reduce network load. There are two types of servers in CDN compositions: origin and replica servers (so-called surrogate servers) [6]. We can therefore differentiate CDNs on the basis of their surrogate servers placement, and classify them into core and edge architectures. Core CDN architectures rely on the deployment of private datacenters close to ISP points of presence (PoP), and it has been a successful approach used by most of the big

DCN infrastructure providers, including Akamai [5]. Since this approach uses private resources deployment, and are not designed for cooperating with other CDNs, they require huge amounts of money for deployment and maintenance. Yet as core architectures are connected to PoPs, they do not have control of traffic throughout ISP until the end-customer that undermines QoS guarantees enforcement. Interoperable CDNs in edge network have emerged to tackle directly these issues. Network service providers look forward to (i) take advantage of their infrastructure, (ii) deploy their own datacenters, and (iii) deliver content as close as possible to end-customer. The aim is to be able to offer differentiated QoS guarantees to regular customers [8]. In this work, we focus on challenges risen by edge CDN architectures. In particular, we have studied how to organize consumer-edge devices to cooperate with small-sized datacenters, and how to enforce different classes of strict SLA contracts at the edge of the network.

Replication Schemes: Network providers can rely on replication schemes for enhancing disseminating content efficiency. Considering resource allocation strategies, we are mostly interested in two categories of replication schemes: uniform, and adaptive replication schemes. The Google File System (GFS) [4] and Ceph [9] adopt a pragmatic approach where the number of replicas is uniform, that mean a fixed number of replicas per stored object. This trivial and primitive approach has had a considerable success in the industry, particularly for datacenters deployment, because it is easy to adopt. However it relies on over-provision to provide resource allocation for popular content, and despite of using commodity servers, it is inefficient and quite expensive. Overall, these issues are addresses by adaptive replication schemes. For instance, the use of non-collaborative LRU caching allows us to easily adapt the replication degree of an content according to its demand. More sophisticated adaptive schemes, such as EAD [7] and Skute [1] tackle content replication by using a cost-benefit approach over decentralized and structured P2P systems. EAD creates and deletes replicas throughout the query path with regards to object hit rate using an exponential moving average technique. Skute provides a replication management scheme that evaluates replicas price and revenue across different geographic locations. Its evaluation technique relies on equilibrium analysis of data placement. Despite being highly scalable and providing an efficient framework for replication in distributed systems, these approaches result in inaccurate transfer rate allocations, hence they are inappropriate for high-quality content delivery.

3 Approach

3.1 Caju’s Design

We introduce a simple content distribution system, called Caju to study adaptive replication schemes at the edge of the network. Its design is depicted in Figure 1. We assume that the service provider infrastructure is organized in federated storage domains. A storage domain is a logical entity that aggregates a set of storage elements that are located close to each other, e.g. connected to a digital subscriber line access multiplexer (DSLAM). The storage elements are partitioned

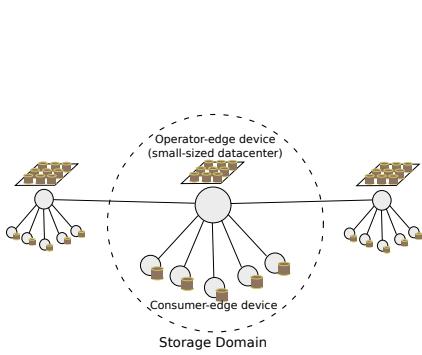


Fig. 1. Storage Elements (SEs) and Storage Domains (SDs)

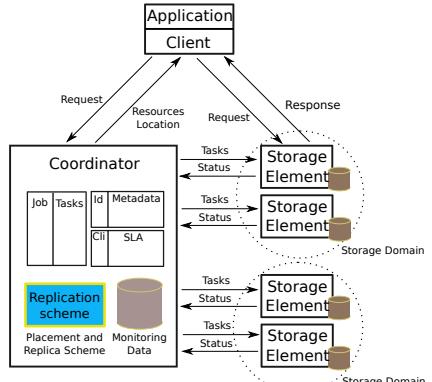


Fig. 2. The main functional blocks

in two different classes: (i) operator-edge elements, furnished by storage operators, e.g. small-sized datacenters, and (ii) consumer-edge entities provided by consumers, such as set-top boxes. Consumer-edge devices contribute to storage and network resources according to their availability and load. Operator-edge nodes run a distributed storage system for local-area network over commodity servers. They provide cheap and high available resources dedicated to the storage service.

On top of each Storage Domain runs a couple of services that deals with serving clients' requests, and performs appropriate object data placement and replication. The main functional blocks are depicted in Figure 2. Remote storage clients contact the coordinator when they need perform any request over an object (PUTs and GETs). The coordinator maintains a catalogue of all clients and available resources, it is also responsible for scheduling the requests. On behalf of the clients, it selects proper resources for fulfilling their SLA contracts based on replications schemes.

3.2 System Model

Here, we formally describe the target storage systems' main components, interactions, and constraints. We also provide guidelines on our performance goals for the evaluation Section 4.

Object Store Service and Client Satisfaction. Our target system provides a distributed object store service for clients. We denote the set of all possible client's objects as \mathcal{O} . We consider that objects comprise a set of data blocks of fixed size, K_C , called chunks. So that, an object $o \in \mathcal{O}$ of size z_o has $\frac{z_o}{K_C}$ chunks.

We consider that M clients are able to do any number of requests R_M to the system. There are three types of client request: *get* any object, *put* new objects it into the system, and *delete* their own objects. The storage system provides a fourth system request type in order to *replicate* an object. It also might perform *deletions*, a fifth request type, for maintenance or control purposes.

We assume that clients are eager for quality of storage service, and their wills are formally defined by SLA contracts. SLAs allow a client m to choose a suitable rate of chunks per request λ_s and minimum acceptable percentage of successful requests P_s . Assuming a period of request analysis T , we consider that a client m who did r'_m requests is satisfied with the store service if at least $r'_m x P_s$ requests were accomplished with λ_s rate. The object store service places and replicates objects throughout the system with regard to the client satisfaction.

Storage at the Edge of Network Providers. We consider a distributed storage system deployed at the edge of network providers, that is organized in storage domains. We assume that there exists I storage domains. A storage domain i , $i \in \{1, 2, \dots, I\}$ has storage capacity of S_i and throughput T_i . Each storage domain has a set \mathcal{J}_i of J storage elements, $j \in \{1, 2, \dots, J\}$, partitioned in two distinct classes: \mathcal{C}_o for operator-edge class, and \mathcal{C}_c for consumer-edge class, where $|\mathcal{C}_c| \gg |\mathcal{C}_o|$. The storage capacity of storage element j is denoted by:

$$s_{ij} = \begin{cases} D_o & \text{if } j \in \mathcal{C}_o; \\ D_c & \text{if } j \in \mathcal{C}_c. \end{cases} \quad (1)$$

where D_o and D_c are maximum storage capacity parameters. Hence,

$$S_i = \sum_{j=1}^J s_{ij} \quad i \in \{1, 2, \dots, I\} \quad (2)$$

The storage element bandwidth capacity is denoted by:

$$b_{ij} = \begin{cases} W_o & \text{if } j \in \mathcal{C}_o; \\ W_c & \text{if } j \in \mathcal{C}_c. \end{cases} \quad (3)$$

where W_o and W_c are maximum bandwidth capacity parameters. We assume that any storage element has a full-duplex, symmetric connection links. Moreover, b_{ij}^u denotes the instantaneous bandwidth consumption of storage element j of i . In a same storage domain i , if j and j' are two storage element from different classes, and there is not active transfer between them, their respective b_{ij}^u do not interfere with each other. Despite that, we consider that network infrastructure imposes the following condition (4) on the maximum throughput of a set of consumer-end storage elements of i :

$$\sum_{j \in \mathcal{C}_c} b_{ij}^u \leq W_l \quad (4)$$

where W_l is a the maximum aggregated bandwidth consumption for a set of consumer-edge devices that the network provider infrastructure permits. Considering inequality (4), the maximum throughput of a storage domain i is denoted as follows:

$$T_i = \frac{1}{K_C} \left(\sum_j b_{ij} \right) \leq \frac{1}{K_c} (W_l + |\mathcal{C}_o| W_o) \quad (5)$$

*where K_C is the chunk size parameter.

System Interactions and Performance Goals. Each client m is connected, through its own consumer-edge storage element j , with a single domain i , called home storage domain. Any m belongs to a SLA class. The system might have one or many SLA classes, such that different levels of quality of service might be provided. As described here above, SLA's constraints allow clients to choose a suitable rate of chunks per request λ_s and minimum acceptable percentage of successful requests P_s , and that the client satisfaction depends on these parameters.

The system allows clients to do storage requests towards their own homes only. However, all requests might be served by storage elements from any federated storage domain, except for objects' insertions, that must be served by client's home storage domain. For mapping and monitoring resources, and interactions in our storage system, we assume that there exists logically centralized coordinator. The performance and design issues of coordinator are beyond the scope of our current work.

We denote the set of all R possible requests by \mathcal{R} . Requests are grouped in two distinct manners: by requester or by type of request. In terms of requester, there are two disjoint subsets: \mathcal{R}_M for client's requests, and \mathcal{R}_S for own storage system's requests. When our system receives a request r that requires to move objects between any node and storage element of i , it serves this request by creating data transfers from a source to a destination. As described above, a requester might be either a client or the own system. If $r \in \mathcal{R}_S$ the transfer is always made between two storage elements. For all $r \in \mathcal{R}$, let:

$$p_{j,r} = \begin{cases} 1, & \text{if } j \text{ serves } r; \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

be a 0-1 variable indicating if the storage element $j \in \mathcal{J}_i$ provides resources to serve request r . We assume there is a function $A_j^i(t)$ that yields the current available rate of chunks by j in t for serving a system incoming request. Therefore, if client m requests r^m over a storage domain i in time t , asking for a λ_m^s rate, the storage system fulfil m 's expectations if and only if:

$$\textbf{Constraint 1: } \sum_{j \in \mathcal{J}_i} p_{j,r}^m \cdot A_j^i(t) \geq \lambda_s \quad (7)$$

Therefore our system performance goals for our replication scheme are twofold. Firstly, we aim to maximize the number of satisfied clients as a metric for evaluating the quality of provided service. And secondly, we tend to minimizing the amount of system's bandwidth and storage usage, by adjusting properly the resource allocation over storage elements in order to serve \mathcal{R} .

4 Evaluation

Our evaluation has two main goals: (i) to evaluate the performance of Caju in providing storage for cloud users on top of edge devices, including operator-edge

devices, so-called small-sized datacenters; (ii) to compare and evaluate challenges of two replication schemes: uniform with fixed number of replicas and non-collaborative caching.

The evaluation scenario (Figure 3) includes 2002 (numbered) nodes arranged across two Storage Domains (SD). There are one operator-edge device (nodes 1 and 1001) and 1000 consumer-edge devices per Storage Domain. Storage and network capacities differ accordingly to the class of device. Each operator-edge device has 10TB of storage capacity and 4Gbps as network capacity. Consumer-edge devices contribute with a smaller storage capacity per device, 100GB, and are equipped with a full-duplex access link of 100Mbps per consumer device. We consider that consumer-edge devices that belong to the same Storage Domain are geographically close to each other, and that a maximum bandwidth limit of 80% is enforced to aggregated traffic of consumer-edge devices on edge network level.

The workload was carefully set-up to match to multimedia popular content distribution, as described in recent studies [3]. Tables 1 and 2 list default values for evaluation scenario and workload parameters respectively. SLA contracts differ to each other by transfer rate λ_s . Thus, we consider three SLA classes, in chunks per second: (a) 41, (b) 21, and (c) 14 chunks/s. To each customer is assigned a SLA that regards the following distribution: 40% class (a), 40% to (b), and the remaining 20% to (c). We assume that a SLA violation occurs when any transfer of a consumer does not observe her minimum contracted transfer rate.

We use *happiness* or number of customers without SLA violations as a key performance metric. This means P_s is equal to 100% in our model. Along with *happiness*. We also focus on number of SLA violations, number of flows, storage and network capacity usage.

The rest of this section is structured as follows. Subsection 4.1 describes the two replication schemes evaluated in this work. Then, we show our performance analysis for these two schemes in Subsection 4.2.

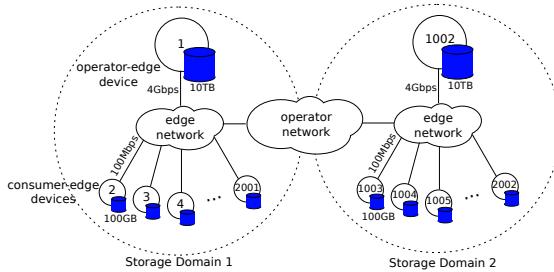


Fig. 3. Evaluation scenario

4.1 Evaluated Replication Schemes

We have evaluated the performance of two replication schemes with Caju:

Uniform Replication Scheme with Fixed Number of Replicas. This is the simplest approach to replicate objects into a system, that is broadly used in

Table 1. Default parameter values for the evaluation system

Evaluation scenario	
Number of Storage Domains (SD)	2
Number of Operator-edge devices (OE) per SD	1
Number of Consumer-edge device (CE) (with a home client) per SD	1000
OE storage capacity	10TB
CE storage capacity	100GB
OE network capacity	4Gbps
CE network capacity	100Mbps
Aggregate bandwidth limit for a set of CEs	80%
Chunk size	2MB
Number replicas	2
Maximum parallel flows per request	5

Table 2. Default values for workload parameters

Workload	
Requests per client	uniform
Experiment duration	1h 12min
Object size (follows Pareto)	shape=5 lower bound=70MB upper bound=1GB (mean 93MB)
Mean requests per second	50
Requests division	5% for PUTs 95% for GETs
Popularity growth (follows Weibull)	shape=2 scale \propto duration
Content popularity (Zipf-Mandelbrot)	shape=0.8 cutoff=# of objects
PUTs (Poisson)	λ =PUTs/s

current datacenter deployments. Given a fixed number of replicas n as a parameter, we simulate a chain of object-replication of n stages just after the initial insertion (PUT). Requests are scheduled in order to balance load throughout nodes. Each request might be served by at most R nodes with equal load. The actual number of sources is $r = \min(n, R)$.

Non-collaborative LRU Caching. Simple adaptive replication schemes based on non-collaborative caching, such as those that implements Least Recent Used algorithm, are straightforward and easy to implement and deploy. In our implementation, a new replica is created whenever a client, connected to a operator-edge device, performs a GET to any object. LRU replacement is enforced regarding a static percentage of the local storage capacity γ for caching. Request scheduling is quite similar to that of uniform approach. However the number of available sources n changes according to LRU algorithm.

4.2 Performing Storage for Cloud Users at the Edge of the Network

We analyse the efficiency of delivering popular content with strict SLA definitions using two replication schemes approaches: uniform replication with a fixed number of replicas, and non-collaborative LRU caching.

First, we have evaluated the required number of replicas of uniform replication for different request rates in order to prevent SLA violations. Figure 4 shows *happiness* metric for mean request rates of 50, 100, 150, and 200 requests per second. We have observed that uniform replication schemes require high replica-

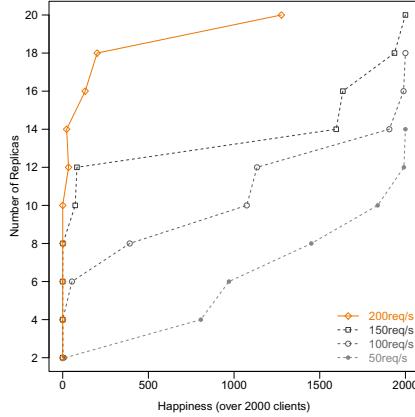


Fig. 4. Happiness metric with uniform replication scheme

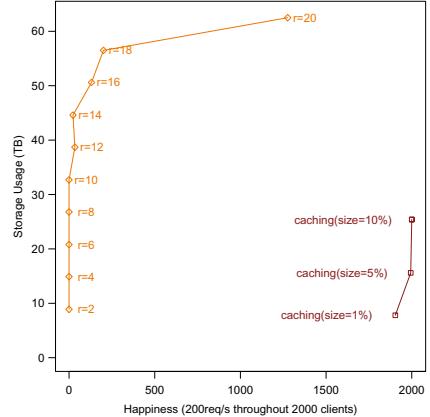


Fig. 5. Storage usage for uniform and caching replication schemes

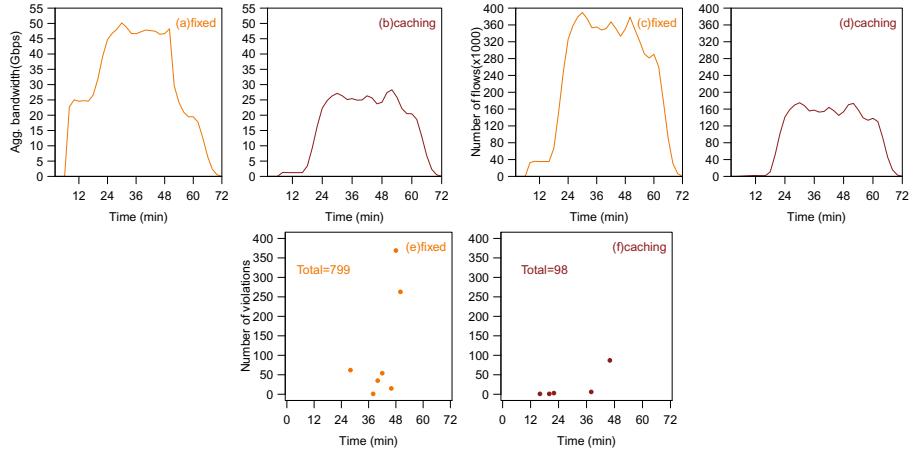


Fig. 6. Aggregate bandwidth (a,b), number of flows (c,d) and number of SLA violations (e,f) using fixed number of replicas and caching.

tion degree in order to cope with strict SLA definitions and popular content. At least 20 replicas are required to prevent violations if the request rate is as high as 150 requests per second. For the highest request rates, uniform replica is not suitable. When we simulated 200 requests per second, there were 799 violations. Despite having been widely used in datacenters and storage clusters, uniform replication scheme relies on over-provision in order to distribute popular content with strict SLA definitions, hence it is not fit for edge network deployments.

To avoid over-provision, we have analysed the storage usage uniform replication with a non-collaborative LRU caching. We simulate different LRU caching sizes percentages: 1%, 5%, and 10% of the storage capacity. Figure 5 plots stor-

age storage usage and *happiness* metric for 200 requests per second. Even with the smallest cache storage percentage of 1%, a non-collaborative LRU caching approach performs much better than uniform replication. We observed 89 violations with 1% of non-collaborative LRU caching that required a storage usage, 7.82TB, similar to uniform scheme with 2 replicas, 8.92TB.

In order to gather more information about the advantages of using non-collaborative caching for distributing popular content instead of uniform replication, we have evaluated and plotted in Figure 6 the throughput, flows, and violations results sampled per second. We selected results from LRU caching with local cache size of 1% of the node storage capacity, and uniform replication with 20 replicas. By using a non-collaborative LRU caching, we have seen that the number of flows and aggregate bandwidth was reduced by half. We have also verified that the number of violation slashed from 799 to only 98.

5 Conclusions and Perspectives

Online storage of big data becomes very popular. Storage providers need to find good trade-offs between replication and storage usage. In this paper, we show that it is important to take content popularity into account: using a fixed replication would lead either to waste storage space or to increase the number of unsatisfied customers. We propose and evaluate Caju, a content distribution system for edge networks. Caju provides the ability to manage storage and network resources from both consumer and operators in a collaborative manner. Our evaluations show that non-collaborative caching consistently outperforms the fixed replication scheme. It provides a eight-fold decrease in the number of SLA violations, requires up to 10 times less of storage capacity for replicas, and reduces aggregate bandwidth and number of flows by half. We are working on the design of new adaptive placement and replication algorithms. Our goal is to enhance non-collaborative caching for popular content delivery.

References

1. Bonvin, N., Papaioannou, T.G., Aberer, K.: A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In: ACM (ed.) ACM Symposium on Cloud Computing 2010, SOCC 2010, Indianapolis, USA (June 2010)
2. Cisco visual networking index: Forecast and methodology (2010-2015), <http://www.cisco.com> (June 2011)
3. Figueiredo, F., Benevenuto, F., Almeida, J.M.: The tube over time: characterizing popularity growth of youtube videos. In: Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, pp. 745–754. ACM, New York (2011)
4. Ghemawat, S., Gobioff, H., Leung, S.-T.: The google file system. In: SOSP 2003: Proceedings of the 9th ACM Symposium on Operating Systems Principles, pp. 29–43. ACM Press, New York (2003)
5. Nygren, E., Sitaraman, R.K., Sun, J.: The akamai network: a platform for high-performance internet applications. SIGOPS Oper. Syst. Rev. 44(3), 2–19 (2010)

6. Pathan, M., Buyya, R.: A Taxonomy of CDNs. In: Buyya, R., Pathan, M., Vakali, A. (eds.) Content Delivery Networks. LNEE, vol. 9, pp. 33–77. Springer, Heidelberg (2008)
7. Shen, H.: An efficient and adaptive decentralized file replication algorithm in p2p file sharing systems. IEEE Transactions on Parallel and Distributed Systems 21, 827–840 (2010)
8. Enabling digital media content delivery: Emerging opportunities for network service providers (March 2010), <http://www.velocix.com/formwp.php>
9. Weil, S., Brandt, S.A., Miller, E.L., Long, D.D.E., Maltzahn, C.: Ceph: A scalable, high-performance distributed file system. In: Proceedings of the 7th Conference on Operating Systems Design and Implementation, OSDI 2006 (November 2006)

Data Security Perspectives in the Framework of Cloud Governance

Adrian Copie^{1,2}, Teodor-Florin Fortiș^{1,2}, and Victor Ion Munteanu^{1,2}

¹ Institute e-Austria, Timișoara
bvd. V.Pârvan 4, Timișoara, Romania

² West University of Timișoara,
Faculty of Mathematics and Informatics,
Department of Computer Science
bvd. V.Pârvan 4, Timișoara, Romania
`{adrian.copie,fortis,vmunteanu}@info.uvt.ro`

Abstract. The adoption of Cloud Computing paradigm by the Small and Medium Enterprises allows them to associate and to create virtualized forms of enterprises or clusters that better sustain the competition with large enterprises sharing the same markets. In the same time the lack of security standards in Cloud Computing generates reluctance from the Small and Medium Enterprises in fully move their activities in the Cloud. We have proposed a Cloud Governance architecture which relies on mOSAIC project's cloud management solution called Cloud Agency, implemented as a multi-agent system. The Cloud Governance solution is based on various datastores that manage the data produced and consumed during the services lifecycle. This paper focuses on determining the requirements that must be met by the various databases that compound the most complex datastore from the proposed architecture, called Service Datastore, together with emphasizing the threats and security risks that the individual database entities must face.

Keywords: Cloud Computing, Cloud Governance, Datastores, Databases, Security in the Cloud.

1 Introduction

Recently, Cloud Computing has become an omnipresent paradigm which offers a plethora of advantages and opportunities from the technological and economical points of view. It has the potential to revolutionize the way we see and do business by providing clear economic incentives (e.g. the *pay per use* economic model) [1, 2]. Its approach on selling and renting cloud resources and services is enabled through essential characteristics like on-demand self-service, broad network access, resource pooling (multi tenancy), rapid elasticity, measured services, service orientation, strong fault tolerance and loosely coupled services [3–5].

The diversity in proprietary technologies deployed by the different cloud vendors has caused what is known as *vendor lock-in*, slowing down the adoption

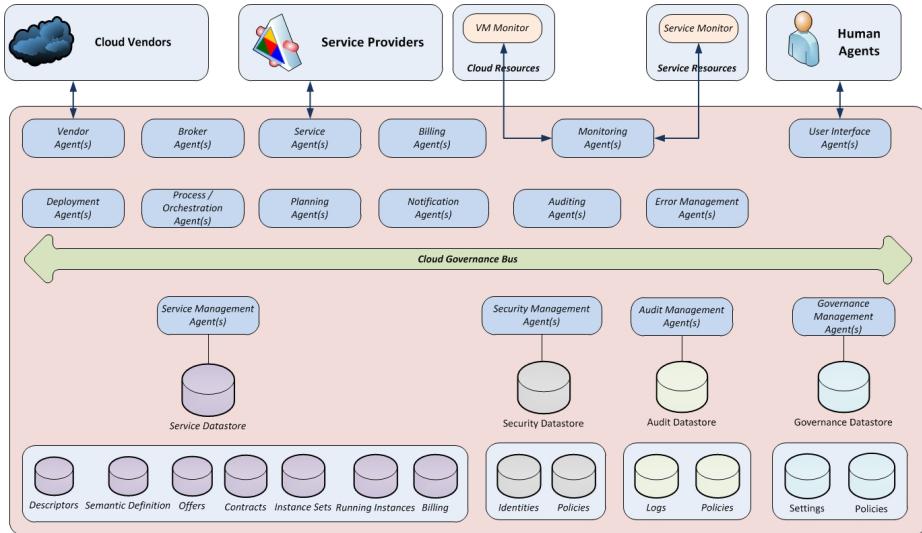


Fig. 1. Multi-Agent Governance Architecture

process. Different platform-as-a-service (PaaS) solutions, including mOSAIC¹, Cloud Foundry², OpenShift³, Morfeo 4CaaSt⁴, ActiveState's Stackato⁵, WSO2 Stratos⁶, attempt to mitigate this problem.

Available PaaS solutions have enhanced cloud adoption by allowing the development of multi-cloud services but, because of the lack of features of current PaaS solutions like service lifecycle management, service discovery, contract brokering, and others, there is a high degree of fragmentation, most of the services being run rather in isolation [6].

Cloud adoption is an ongoing process which gives to the SMEs the opportunity to develop highly specialized solutions or to cooperate and work together as a virtual enterprise in order to offer complex solutions, tailored for their customers' needs. To build effective solutions inside virtual enterprises, it is necessary to implement appropriate cloud management and cloud government policies [7] acting together as distinct components of a cloud infrastructure.

In the papers from Distributed Management Task Force (DMTF, [8, 9]) there are emphasized the cloud management and cloud governance requirements, having them in close relation and describing their roles, which allow us to propose a security oriented cloud governance architecture.

Our paper focuses on security aspects regarding lifecycle support for the proposed cloud governance architecture and it is structured as follows.

¹ <http://www.mosaic-cloud.eu/>

² <http://cloudfoundry.org/>

³ <https://openshift.redhat.com/app/>

⁴ <http://4caast.morfeo-project.org/>

⁵ <http://www.activestate.com/stackato>

⁶ <http://wso2.com/cloud/stratos/>

The results of our paper are covered in Sections 2 and 3. Finally, Section 4 draws conclusions and discusses future work.

2 Governance Datastores

A multi-agent cloud governance architecture, as depicted in Figure 1, targets service management, security and privacy management, service life-cycle management and monitoring by using agent technologies. It is composed of four interdependent subsystems: *Service Management*, *Security Management*, *Audit Management* and *Governance Management*.

This architecture was designed with several issues in mind:

- Compliance with business standards;
- Complete service management (registering, updating, searching, removing);
- Automated service lifecycle (instantiation/commissioning, contracting, retirement etc.);
- Security and privacy management that is compliant with government laws.

The cloud management component relies on mOSAIC’s Cloud Agency, implemented in the framework of the ‘Open Source API and platform for multiple Clouds’ (mOSAIC) project. The Cloud Agency is a multi-agent system that performs resource provisioning, monitoring and reconfiguration and is accessible through a REpresentational State Transfer (REST) interface [10–12].

Table 1. Components of the Service Datastores

Component	Description	Stored data type	Implemented as
Service Descriptor	Functional and non-functional requirements for the services	Structured, searchable	RDBMS
Semantic Definitions	Semantic description of the services	Formatted data, searchable through SPARQL	RDF datastore
Offers	Describes the service parameters and the price	Unstructured information	Graph database
Contracts	The agreement between the service provider and service consumer	Unstructured information	Graph database
Instance Sets	Aggregation of trading, billing and deploying information	Unstructured information	Graph database
Running Instances	Information about the live service instances	Unstructured information	Graph database
Billing	Billing information generated as a result of service usage	Unstructured information	Graph database

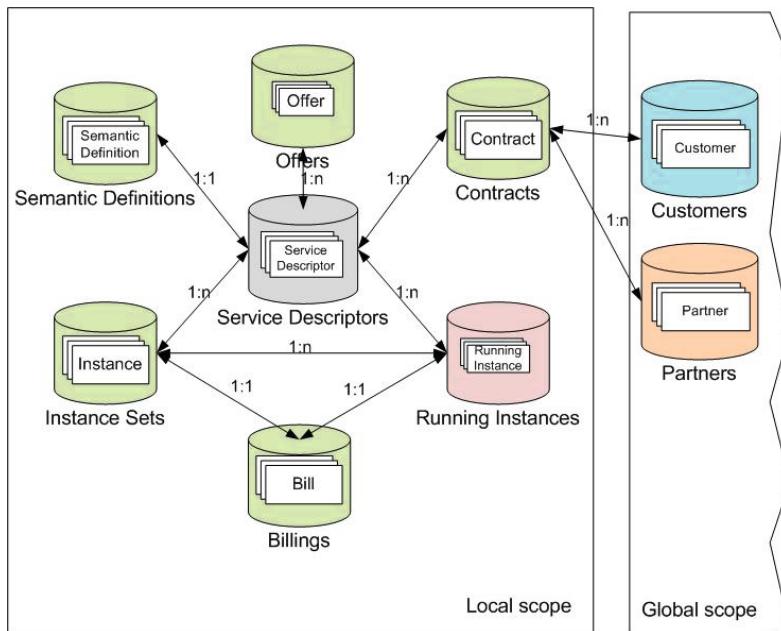


Fig. 2. Service Datastore

The information persisted inside the Cloud Governance system is extremely heterogeneous due to the fact that every agency manipulates different data structures, leading to a complex mesh of databases. By data integration through Enterprise Integration Patterns, all the information is combined and presented to the governance component in a consumable format. The Cloud Governance Bus (as identified in Figure 1) allows the messages exchange and the data aggregation through the agencies related to the different datastores. More than being a complex storage in terms of number of databases, also their types are varying in order to offer best performances for specific operations.

2.1 Service Management Datastore

This storage system has the role to maintain a coherent information about all the services registered inside the Cloud Governance environment. The information stored in this datastore describes the services from syntactic and semantic point of view, but it is also needed by other processes like offers consulting, contract establishment, billing generation or maintaining accurate data about the running services instances.

Because Cloud Governance is an evolution of the Service Oriented Architecture (SOA) governance, it relies also on a Service Repository model involving methods called by the providers to register their services and methods called by the consumers in order to discover the existing services. The structure and the content of the information inside the Service Datastore is grouped in several

components described in Table 1 along with the data types they store and also their practical implementation in terms of used database type. Figure 2 depicts the architecture and the relationship between all the components of the Service Datastore. All these components are interdependent, they have to be kept synchronized and up to date all the time to provide accurate data. More than this they are related also with two global catalogues namely *Customers* and *Partners* used in the same time by other Cloud Governance modules.

2.2 Security Management Datastore

The access to cloud services must be performed in a secured way, only authorized parts having the rights to use them fully or in a granular way, based on specific access control lists or policies. The Security Manager agent is responsible for the credentials management, the authentication and authorization processes and the generation of the security tokens that will be further used in accessing the resources by the other agents in the Cloud Governance environment.

All the credentials and security policies governing the cloud services are stored inside the Security Datastore. The identities of all the service consumers together with their representative credentials like passwords and secret keys used to access specific resources are kept in the Security database. This information must be optimized for reading and also is required to be highly searchable and fast, in order to speed up the authentication process, our approach being a RDBMS database.

2.3 Configuration Management Datastore

Every service provider has its policies and constraints and have to publish them in the services catalogue in order to be taken into account during the service execution phase. The information related to the policies and constraints is unstructured due to the multitude of service types that could cooperate. The data is usually related to the service functionality, guaranteed parameters of services, security policies, configuration parameters of the virtual machines on which agents are executed and many more.

In the absence of the governance policies or the functional constraints, the governance process can not happen, so this is a critical repository which requires high availability and reliability. Also it must be searchable and editable which finally lead us to a graph database.

2.4 Monitoring and Audit Management Datastore

The current cloud management solution based on mOSAICs Cloud Agency is not enough in order to monitor the registered cloud services because it acts at the infrastructure level. This is why the Cloud Governance system is responsible for this task through its Audit and Monitoring Management agent. The Audit Datastore must face to a high number of writes since all the services taking part in the cloud governance system are continuously generating data related to audit actions or information resulted from the monitoring process. The goal is

to provide a highly available for writes datastore. All the data generated as the result of auditing or monitoring are stored as they come, the only operation at this phase is the insertion in the database.

In the same time, this datastore must offer the possibility of analysing the data and performing different predefined actions based on the result of the data analysis (e.g. intrusion detection, insufficient application resources, stopped services, etc). The nature of the stored data, the number of records and the complexity of the generated information qualify this datastore to be considered as a Big Data store. The data is unstructured and the volume depends basically on the number and type of monitored service parameters like the number of the connections to a database or the number of elements in a message queue at a specific moment, the sampling frequency and the number of actions that require auditing. Special strategies need to be defined in order to deal with the content of this storage, to search and process the data and produce valuable results.

Because the data volume is high, the store have to support scalability to accommodate with the amount of generated data. Also the store must be highly available for writes and must easily scale horizontally, that is the reason why we have chosen a key-value datastore for our implementation.

3 Requirements for Service Datastore Security

Security is the most important concern that prevents SMEs to fully adopt Cloud Computing. The Cloud Governance system handles extremely sensitive data belonging to the SMEs and their partners, therefore the security requirements are very strict and they must be implemented and respected regardless the functional component in the governance environment. The Service Datastore has a complex structure built from many different databases holding different data types. Every database type has inherent security issues that must be first understood and then addressed individually. In the following sections, the particular security threats will be discussed and solutions in order to mitigate the possible risks are proposed.

3.1 Service Datastore

Services Repository. This repository has to be highly readable as it must face a large number of requests related to the discovery process. The number of writes is smaller than the number of reads since new services registration occurs much seldom than usual interrogations.

The Service Descriptors must provide searchability, which must be enough flexible to respond to various search criteria. These requirements lead to a RDBMS solution which can be implemented in the public cloud through many existent NewSQL solutions, as well in the private cloud by using a relational database optimized for reading.

This kind of storage comes with some possible threats and risks that must be taken into account at the implementation phase. Wherever the database would

be (in the public or private cloud), it is not directly exposed to attacks, but indirectly through carefully chosen malicious input. The best known attack of this type is the SQL injection which acts to the Data Manipulation Language (DML) as well to the Data Definition Language (DDL) and could alter or even destroy the entire database content or even compromise the entire service.

Semantic Definitions. The data items contain both data itself together with their representation schema and the XPath queries contain the values to be queried for and the query itself. Sometimes, this data and code mixture could be exploited by malicious text chunks intercalated among the XML items, just like in the SQL injection technique.

The XML parsers used to access the RDF repositories are also a source of possible security risks due to their vulnerabilities to various exploits. For example the SAX parsers are less prone to vulnerabilities due to the fact that they usually don't support the full XML implementations. However their serial reading, with data superseding, leads to less control over the invalid or unclosed XML.

Using the same analogy with the SQL injection, the prevention for the XML injection consists in a better input validation, looking for specific code patterns, checking not only the data type itself but other characteristics like format, length and content. Data validation at both endpoints is also required in order to be sure that the code that reach the server has no malicious potential.

The RDF stores are implemented as in-memory databases, native disk storages or support relational databases plugins. The users management for these persistence approaches could bring supplementary threats over the Service Data-store and they should be seriously treated at multiple levels.

Offers, Contracts, Instances, Running Instances and Bills Databases. The graph database, like other NoSQL databases, suffers from different kinds of risks and vulnerabilities that must be understood and properly combated. This type of database has a high level of horizontal scalability, so it carries with it the issues encountered in the distributed systems like connection pollution and Man-in-the-middle (MITM) attacks.

Even if the installation is behind a firewall, these risks are present and various defending techniques like Public Key Infrastructures (PKI), stronger mutual authentication between nodes or second channel for verification should be added. Because the information inside those databases is in some cases formatted using XML schemas, data manipulation is also a potential risk.

Every database support inputs from different components, which leads to supplementary data validation procedures to enhance the security and mitigate code injections. Many graph databases does not come with an individual authentication mechanisms, relaying in most cases on the host operating system user management. This is usually not enough because of the granularity lack, most permission being at the file level. The applications using the database often have to deal with the Access Control Lists or other access policies.

Table 2. Security Controls that apply to the Service Datastore

Security Control	Description
Assets Management	Service Datastore could reside partially in the private and in the public cloud. However, for the databases that are hosted in the private cloud it must be possible to manage all the infrastructure components (physical or virtual machines, networks, software)
Data/Storage Security	It must be possible to store data in an encrypted format. In the same time, it must be possible to store data in separate locations as special requirements came from the service consumers.
Endpoint Security	Every endpoint composing the Service Datastore must be fully securable, including restrictions related to protocols and access devices.
Network Security	It must be possible to secure the traffic at the switch, router and packet levels. Also if the public cloud is selected to host various storage systems, appropriate cloud providers must be chosen, that can make the proof of an adequate security support.
Workload and Service Management	The services must be configured, deployed and monitored according to defined security policies and customer license agreements

Table 3. Security Patterns that apply to the Service Datastore

Security Pattern	Description
Identity Management	Having many database components as building blocks of the Service Datastore and taking into account that they must communicate permanently, an identity management mechanism is required in order to provide security tokens that will be used internally, wherever necessary, to prove the requester's identity.
Access Management	The security tokens must be examined and determined what level access is allowed over a specified resource.
Configuration Management	The databases compounding the Service Datastore could be installed in physical or virtual configurations, running on-premise or in cloud. It is necessary a mechanism that is able to federate this configuration data and offer it as an aggregated information in order to be used across multiple domains.

Depending on the access level of a hypothetical attacker, at the database level or at the file-system level, the records could be corrupted, destroyed or stolen, the worst scenario being the corruption or destruction of the entire database.

3.2 Security Controls and Patterns in Service Datastore

According to [13], different security controls are necessary to secure the Cloud Governance system. This environment relies on different specialized databases

and the Service Datastore is the most sophisticated. Due to its complexity, various cloud security controls could be applied. We have identified some of them, namely *Access Management*, *Data/Storage Security*, *Endpoint Security*, *Network Security*, *Workload and Service Management*.

Table 2 provides a synthesis of these security controls and details their connections to the concrete Service Datastore architecture. Every identified control is described from its functional point of view.

Service Datastore is a complex mesh of databases being seen like a federation of various resources for which we have identified different security patterns in conformity with [13]. The Service Datastore components could reside in the private cloud but also in the public cloud. This is why strong security requirements must be fulfilled to assure an appropriate protection of governance information together with sensitive data belonging to the consumers of the registered cloud services. Table 3 presents some of the security patterns identified inside the Service Datastore component.

4 Conclusions and Future Work

Cloud Computing adoption allows SMEs to access new markets where they can associate in virtual enterprises or virtual clusters, being able to better sustain the competition with the large enterprises acting on the same markets. However, the lack of standardization in what concerns the Cloud Computing security prevents the SMEs to fully embrace the cloud technologies and to move their activities entirely in the cloud.

We have proposed a Cloud Governance architecture that relies on the mOSAIC's Cloud management solution that aims to solve the management and governance of the services infrastructure. The proposed governance solution is based on various datastores that manage the data produced and consumed during the life cycle of the cloud services.

Our paper focused on the most complex datastore in our governance system, the Service Datastore, analysing the requirements related to the databases type selected to hold all the data produced and consumed during the services lifecycle. In the same time we have pointed the security threats and risks facing the various databases and also proposing solutions to mitigate them. This paper is a base for our future work consisting in implementing the security mechanisms in our proposed Cloud Governance architecture.

Acknowledgments. This work was partially supported by the grant of the European Commission FP7-ICT-2009- 5-256910 (mOSAIC), FP7-REGPOT-CT-2011-284595 (HOST), and Romanian national grant PN-II-ID-PCE-2011-3-0260 (AMICAS). The views expressed in this paper do not necessarily reflect those of the corresponding projects consortium members.

References

1. Weinhardt, C., Anandasivam, A., Blau, B., Borissov, N., Meinl, T., Michalk, W., Stößer, J.: Cloud Computing—A Classification, Business Models, and Research Directions. *Business and Information Systems Engineering*, BISE 1(5), 391–399 (2009) ISSN: 1867-0202
2. Weinhardt, C., Anandasivam, A., Blau, B., Stößer, J.: Business Models in the Service World. *IEEE IT Professional*, Special Issue on Cloud Computing 11(2), 28–33 (2009) ISSN: 1520-9202
3. Mulholland, A., Pyke, J., Fingar, P.: Enterprise Cloud Computing: A Strategy Guide for Business and Technology Leaders. Meghan-Kiffer Press, Tampa (2010)
4. Gong, C., Liu, J., Zhang, Q., Chen, H., Gong, Z.: The characteristics of cloud computing. In: ICPP Workshops, pp. 275–279 (2010)
5. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology (September 2011), <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
6. Wainewright, P.: Time to think about cloud governance (August 2011),
[http://www.zdnet.com/blog/saas/
time-to-think-about-cloud-governance/1376](http://www.zdnet.com/blog/saas/time-to-think-about-cloud-governance/1376)
7. ISACA: Cobit 5 introduction (February 2012),
<http://www.isaca.org/COBIT/Documents/An-Introduction.pdf>
8. DMTF: Architecture for managing clouds (June 2010),
[http://dmtf.org/sites/default/files/standards/
documents/DSP-IS01021.0.0.pdf](http://dmtf.org/sites/default/files/standards/documents/DSP-IS01021.0.0.pdf)
9. DMTF: Use cases and interactions for managing clouds (June 2010),
[http://www.dmtf.org/sites/default/files/standards/documents/
DSP-IS0103_1.0.0.pdf](http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0103_1.0.0.pdf)
10. Venticinque, S., Negru, V., Munteanu, V.I., Sandru, C., Aversa, R., Rak, M.: Negotiation policies for provisioning of cloud resources. In: Proceedings of the 4th International Conference on Agents and Artificial Intelligence, pp. 347–350. SciTePress (February 2012)
11. Venticinque, S., Aversa, R., Di Martino, B., Rak, M., Petcu, D.: A Cloud Agency for SLA Negotiation and Management. In: Guerraccino, M.R., Vivien, F., Träff, J.L., Cannatoro, M., Danelutto, M., Hast, A., Perla, F., Knüpfer, A., Di Martino, B., Alexander, M. (eds.) Euro-Par 2010 Workshop. LNCS, vol. 6586, pp. 587–594. Springer, Heidelberg (2011)
12. Aversa, R., Di Martino, B., Rak, M., Venticinque, S.: Cloud agency: A mobile agent based cloud system. In: Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2010, pp. 132–137. IEEE Computer Society, Washington, DC (2010)
13. Cloud Computing Use Cases Group: Cloud computing use cases white paper (July 2010)

CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing – CGWS2012

Frédéric Desprez¹, Domenico Talia², and Ramin Yayhapour³

¹ INRIA and ENS Lyon, France

² University of Calabria, Italia

³ Georg-August University of Göttingen, Germany

CoreGRID is a European research Network of Excellence (NoE) that was initiated in 2004 as part of the EU FP6 research framework. CoreGRID partners, from 44 different countries, developed theoretical foundations and software infrastructures for large-scale, distributed Grid and P2P applications. An ERCIM sponsored CoreGRID Working Group was established to ensure the continuity of the CoreGrid programme after the original funding period of the NoE. The working group extended its interests to include the emerging field of service-based cloud computing due to its great importance to the European software industry. The working group's main goals consist in i) sustaining the operation of the CoreGRID Network, ii) establishing a forum encouraging collaboration between the Grid and P2P Computing research communities, and (iii) encourage research on the role of cloud computing as a new paradigm for distributed computing in e-Science.

In particular, the ERCIM CoreGRID working group managed to organize an annual CoreGRID workshop, traditionally associated to the Euro-Par conference, thus continuing the successful tradition of the annual CoreGRID workshops during the initial Network of Excellence. Past ERCIM CoreGRID workshops have been organized in Delft (2009), Ischia-Naples (2010), Bordeaux (2011). In 2012 the workshop has been organized in Rhodes Island, Greece. The topics of interest included Service Level Agreements, Data & Knowledge Management, Scheduling, Virtual Environments, Network Monitoring, Volunteer Computing Systems, Trust & Security, Self-* and adaptive mechanisms, Advanced programming models, IaaS, PaaS and SaaS, Tools and Environments for Application Development and Execution.

The 2012 ERCIM CoreGRID workshop was organized jointly with the first workshop in Big Data Management (BDMC) and gathered around 40 researchers from the European community, in August 2012. Six papers were presented after a keynote talk from Frédéric Suter (CC-IN2P3, France) about the SimGrid simulation framework.

The first paper presents an evaluation of a distributed storage plugin for Cuminus, an S3-compatible open-source Cloud service over Grid'5000. The results show that the application, while managing big chunks of data, is able to scale with the size of data and the number of processes. The second paper describes the extensions added to FastFlow, a structured parallel programming framework targeting shared memory multi-core architectures, to support the execution of

programs structured as fine grain parallel activities running on a single workstation. Experiments are conducted over state-of-the-art networked multi-core nodes. The third paper studies the management of pipeline workflow applications that are executed on a distributed platform with setup times. Based on a theoretical study of the problem, the authors provide an optimal algorithm for constellations with identical buffer capacities. For the problem with non-fixed buffer sizes, a $b/(b+1)$ -approximation algorithm is presented. The fourth paper presents the design and implementation of a meteorological application using the ASKALON environment comprising graphical workflow modeling and execution in a Cloud computing environment. The experiments show that a good speedup can be obtained when executed in a virtualized Cloud environment with important operational cost reductions. The next paper describes a workload archive acquired at the science-gateway level. Its added value is demonstrated on several case studies related to user accounting, pilot jobs, fine-grained task analysis, bag of tasks, and workflows. Results show that science-gateway workload archives can detect workload wrapped in pilot jobs, improve user identification, give information on distributions of data transfer times, make bag-of-task detection accurate, and retrieve characteristics of workflow executions. The sixth and last paper proposes a novel mechanism for energy adaptation in P2P file sharing protocols to enhance the possibility of a client completing the file download before exhausting its battery. The proposed mechanism can be implemented in BitTorrent. An analysis is given through a comprehensive set of simulations.

We wish to thank all who contributed to the success of the workshop: authors submitting papers, invited speakers, colleagues who refereed the submitted papers and attended the sessions, and Euro-Par 2012 organizers whose invaluable support greatly helped in the organisation of the Workshop.

October 2012

F. Desprez, D. Talia, R. Yayhapour

Evaluating Cloud Storage Services for Tightly-Coupled Applications

Alexandra Carpen-Amarie¹, Kate Keahey², John Bresnahan², and Gabriel Antoniu¹

¹ INRIA Rennes - Bretagne Atlantique / IRISA, France

² Argonne National Laboratory, USA

Abstract. The emergence of Cloud computing has given rise to numerous attempts to study the portability of scientific applications to this new paradigm. Tightly-coupled applications are a common class of scientific HPC applications, which exhibit specific requirements previously addressed by supercomputers. A key challenge towards the adoption of the Cloud paradigm for such applications is data management. In this paper, we argue that Cloud storage services represent a suitable data storage and sharing option for Cloud applications. We evaluate a distributed storage plugin for Cumulus, an S3-compatible open-source Cloud service, and we conduct a series of experiments with an atmospheric modeling application running in a private Cloud deployed on the Grid'5000 testbed. Our results, obtained on up to 144 parallel processes, show that the application is able to scale with the size of the data and the number of processes, while storing 50 GB of output data on a Cloud storage service.

Keywords: Cloud computing, data management, Cloud storage service, HPC applications, Nimbus, Cumulus.

1 Introduction

Important academic and industrial actors have recently started to investigate Cloud computing, a rapidly expanding paradigm for hosting and delivering on-demand services on a pay-per-use basis. The increasing popularity of the Cloud computing model has drawn the attention of the high-performance computing community, research in this area focusing on the usage of Cloud infrastructures as alternatives to the traditional dedicated supercomputers. Tightly-coupled scientific applications have a unique range of computing needs, from scalability and processing power to efficient parallel I/O and high-performance network interconnects. Several studies [7, 8, 11, 20] have analyzed the impact of porting such applications on Cloud environments, evaluating the tradeoff between the benefits of on-demand computing power and the Cloud cost model, and the performance overhead introduced when hosting HPC applications in the Cloud.

However, past research mainly focused on performance as a means to quantify the HPC capability of public [21] or private [7] Clouds. Another key challenge that can directly impact the effectiveness of using clouds for HPC workloads is data management. Many scientific applications generate large amounts of data which need to be persistently stored and made available for further analysis. Typically, supercomputers rely on parallel file systems specifically tuned for tightly-coupled application workloads for

their storage needs. The versatility of Cloud platforms allows users to customize their virtual machines (VMs) with any file system, this solution being adopted by most of the existing studies. Although it allows users to recreate the original environment used for supercomputers, this approach comes with a performance penalty caused both by the time needed to deploy and configure their own storage mechanisms and the unreliability of VMs local disk storage.

In this paper we analyze an alternative solution: providing Cloud storage services for HPC applications executed on Clouds. We investigate the requirements of such a Cloud data management system and we evaluate a tightly-coupled scientific application with a customized open-source Cloud service. Whereas most studies [5, 16, 21] focus on public Cloud infrastructures such as Amazon’s Elastic Compute Cloud (EC2) [2] and its Simple Storage Service (S3) [17], we perform our evaluations on top of an open-source Cloud that enables us to thoroughly control the physical infrastructure and the Cloud configuration. To this end, we rely on the Nimbus Cloud framework [13] and its S3-compatible storage service called Cumulus [3].

The remainder of this paper is structured as follows. Section 2 describes the motivation of our research. In Section 3 we introduce the applications we target and we detail Cloud Model 1 [6], an MPI-based applications that simulates atmospheric phenomena. Section 4 is dedicated to our Cloud storage service solution relying on Cumulus and BlobSeer [15]. An experimental evaluation of this system is presented in Section 5 and finally, Section 6 draws conclusions and ideas for future work.

2 Motivation

Infrastructure-as-a-Service Clouds typically provide the user with a set of virtual machine instances that can host applications. Such VMs are equipped with local disks the applications can use to store generated or input data. This storage solution, however, is not persistent, as the disk is wiped out each time a virtual machine lease ends. To cope with this issue, Amazon provides services such as the Elastic Block Store (EBS) [1]. Essentially, EBS allows users to attach a virtual disk to each of their VMs, which can then be backed up onto S3 to persistently store saved data. This solution has however a major drawback: each EBS disk corresponds to a specific virtual machine, and therefore the various VMs cannot share the stored data. Furthermore, as computing VMs carry out simulations and generate output data on local EBS disks, no application can access the whole final results without scanning each EBS disk and possibly copying the data onto a single disk to enable further processing. In contrast, uploading generated data directly into a Cloud storage system might overcome the limitations of this approach, enabling users to achieve not only persistency, but also the capability to have a globally shared view of their results.

The works that studied the performance of HPC applications in Cloud settings have typically entrusted data storage and management tasks to parallel file systems, in an attempt to recreate the original environments deployed on supercomputers. While this approach has the advantage of providing the application with a standard file system interface, aggregating the local storage space of virtual machines within a distributed file system does not guarantee data persistency. The computed results are either lost at the end of the VM lease, or the user has to manually save them into a persistent

repository, such as Amazon S3, to make them available to higher-level applications. This operation increases the completion time and consequently, the cost of running the application in the Cloud. Additionally, VM failures can lead to data loss and require application re-execution.

Moreover, such applications typically generate several output datasets, one for each intermediate time step of the simulation. These results serve as an input for higher-level tools. For instance, data-mining and visualization tools, such as VisIt [19], may perform real-time data analysis, debugging or data aggregation for visualizing the output at each timestep. Replacing local storage with a Cloud-hosted service may enable real-time visualization and analysis for each dataset, as well as availability guarantees and standard interfaces to facilitate access to data.

3 Application Model

3.1 Tightly-Coupled Application Model

We target tightly-coupled, high-performance computing applications specific to the scientific community. Such applications exhibit a set of common features, discussed below.

Parallel processes. A wide range of HPC applications split the initial problem into a set of subproblems. Then, these smaller subproblems are spread across a fixed set of processes, which handle the data in parallel. Such applications typically rely on message-parsing systems (e.g., MPI) for inter-process communication and synchronization.

Compute-intensive simulations. We consider applications that simulate complex phenomena in various contexts, such as high-energy physics or atmospheric simulations. They usually require significant computing resources and spend more time to compute results than to perform I/O operations.

Massive output data. Real-life simulations involve large-sized output, as they compute a set of variables describing the evolution in time of the modeled phenomenon. They are typically designed to store results and additional application logs in a parallel file system, such as GPFS [18] or PVFS [14].

No concurrent access to files. Each process computes a subset of the problem output data. Concurrently dumping results into a single shared output file may lead to I/O bottlenecks prone to decrease the overall performance of the application. Therefore, we consider applications involving independent processes, which perform write operations in separate files.

3.2 Case Study: The CM1 Application

Cloud Model 1 (CM1) [6] is a three-dimensional, time-dependent numerical model designed for atmospheric research, in particular for modeling major phenomena such as thunderstorms. CM1 simulates a three-dimensional spatial domain defined by a grid of coordinates specified in a configuration file. For each spatial point, the application is designed to compute a set of problem-specific variables, including wind speed, humidity, pressure or temperature. A CM1 simulation involves computing the evolution in

time of the parameter set associated with each grid point. To this end, the 3D domain is split along a two-dimensional grid and each obtained subdomain is assigned to its own process. For each time step, all processes compute the output corresponding to their subdomain, and then they exchange border values with the processes that handle neighboring subdomains. The computation phases alternate with I/O phases, when each process dumps the parameters describing its subdomain to the backend storage system. CM1 is implemented in Fortran 95 and the communication between processes relies on MPI [10].

4 Our Approach: A Scalable Cloud Storage Service

4.1 Background

Cumulus [3] is an open-source Cloud storage system that combines efficient data-transfer mechanisms and data-management techniques, aiming at providing data Cloud services in the context of scientific applications. It is designed to support swift data transfers using S3-compatible interfaces. Cumulus features a modular architecture that enables an efficient interaction with external modules. In particular, it is built on top of a storage interface that allows system administrators to customize their service according to their Cloud's requirements. The storage interface decouples the processing of client requests from the actual implementation of the storage backend, making Cumulus a versatile tool that can be adapted to various contexts. The default storage backend shipped with the Cumulus service implements the interaction with a POSIX-compliant file system. It provides support for interconnecting Cumulus with either local file systems or parallel file systems that expose a POSIX interface, such as NFS, PVFS or GPFS.

4.2 Towards a Cloud Storage Backend for Efficient Concurrent Data Transfers

To provide an efficient alternative for traditional file systems, a Cloud data service has to comply with several requirements specific for large-scale applications.

Support for massive files. Scientific applications typically process huge amounts of records, which cannot be hosted in separate, small files. To efficiently store such datasets, data services have to collect all the records into massive files that can reach Terabytes of data in size.

High-throughput concurrent data transfers. Parallel data processing is one of the crucial features that allow scientific applications to accommodate large amounts of data. To enable efficient support for such applications in the Cloud, data-management platforms have to sustain high-throughput data transfers, even under heavy access concurrency.

Fault tolerance. Reliability is a key requirement for Cloud storage, especially in public Cloud environments. It can be achieved by employing fault-tolerant storage backends. Moreover, *data versioning* is a desirable feature in such contexts, as it enables a transparent support for rolling back incorrect or malicious data modifications.

The aforementioned requirements represent the design principles of BlobSeer [15], a concurrency-optimized data-management system for data-intensive distributed applications. BlobSeer specifically targets applications that handle massive unstructured data,

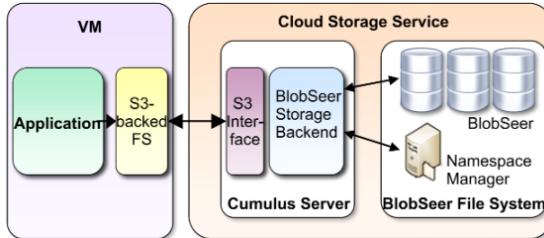


Fig. 1. The architecture of the BlobSeer-backed Cumulus storage service

called *blobs*, in the context of large-scale distributed environments. Its core operations rely on data striping, distributed metadata management and a versioning-based concurrency control mechanism to achieve efficient data transfers under highly-concurrent data accesses. The architecture of BlobSeer and its performance evaluation have been presented in detail in several works, such as [15].

To optimize Cumulus for large-scale, high-throughput concurrent data transfers, we designed and implemented a BlobSeer-based distributed storage backend for the Cumulus service. To this end, we enhanced BlobSeer with a file-system layer, enabling it to meet the requirements of the storage interface defined by Cumulus.

4.3 Design Overview

In order to run unmodified HPC applications in the Cloud and yet benefit from the Cloud storage solutions, we implemented two interface layers. First, we integrated BlobSeer as a backend for Cumulus, by designing a file-system interface for BlobSeer. Second, as typical Cloud storage services expose an S3 interface, we implemented a file-system module to run inside the VMs and stream file-oriented application data to any S3-compatible service. Thus, applications executed inside the VMs store their output data on local paths, assigning the file-system module the task of automatically transferring the data to the Cloud service, i.e. Cumulus in our case, and finally to the persistent storage backend (as shown in Figure 1).

The BlobSeer file-system interface. We designed a file-system layer on top of BlobSeer to enhance it with an easily-accessible and hierarchical file namespace, while preserving the efficient concurrent data operations built into the system. To this end, we equipped BlobSeer with a *namespace manager*, a centralized entity in charge of managing the file hierarchy and mapping files to *blobs*. The namespace manager implements the file system API, exposing the versioning interface of BlobSeer as well as standard file operations, such as create, open, read, write, close. However, the file system layer does not implement the POSIX semantics, preserving instead the original BlobSeer primitives, along with their high-throughput data access guarantees.

The S3-backed file system layer. This module allows applications to interact with standard files that are transparently backed by an S3-compliant Cloud service. This interface layer includes a POSIX-compatible interface relying on FUSE (Filesystem in Userspace) [9]. Each write operation initiated by the application is translated into an upload to the S3 service. As a result, each output file is forwarded to the persistent

Cloud storage and at the same time it is made available to higher-level tools that can process it as the simulation continues. To optimize read operations, we introduced a prefetching mechanism that downloads the files from the S3 repository and stores them locally to improve read access time.

5 Experimental Evaluation

We conducted a set of experiments to analyze the performance and scalability of the Cumulus Cloud storage service in two different contexts. First, we investigated the behavior of the Cumulus service through synthetic benchmarks that assess its data transfer capabilities. Second, we focused on the CM1 application and the impact of the Cloud storage deployment on its performance.

5.1 Environmental Setup

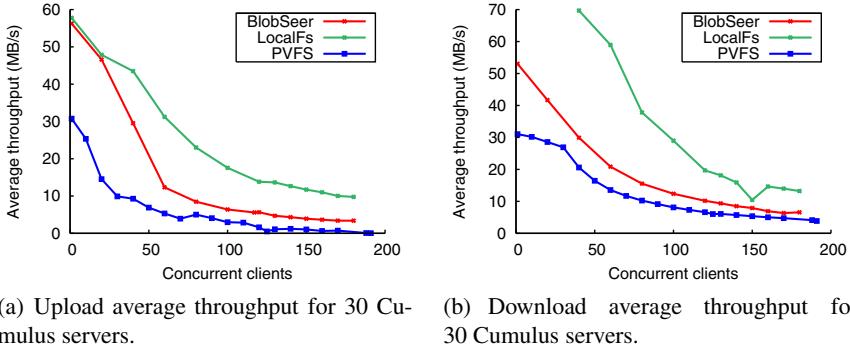
We carried out a set of experiments on Grid'5000 [12], an experimental testbed gathering 10 geographically-distributed sites in France. We used the Rennes cluster of Grid'5000. The nodes it provides are interconnected through a 1 Gbps Ethernet network, each node being equipped with at least 4 GB of memory.

We performed a comparison between several storage backends for Cumulus. First, we used the local disk of each Cumulus server as the storage backend, thus employing the storage space of all the Cumulus nodes. While this approach does not provide a global view of the data and therefore cannot be used for a real-life application, we included this evaluation as a baseline against which to assess the performance of the other backends. The second storage system employed is PVFS [14], a scalable parallel file system designed to provide high performance for HPC applications, by striping data and storing it across several data servers. Finally, we evaluated the BlobSeer-based Cumulus storage. Each experiment involves a distributed deployment configuration, where a set of replicated Cumulus servers use a common storage backend to serve concurrent data transfer requests.

5.2 Cloud Storage Benchmark

To asses the impact of various storage backends on the scalability of the Cumulus service, we performed a set of experiments involving a large number of simultaneous data transfers, so as to simulate a typical HPC scenario when parallel processes concurrently generate output data.

In this experiment, both PVFS and BlobSeer were deployed in a similar configuration, that is 30 data storage nodes and 10 metadata nodes. Replicated Cumulus servers were co-deployed with the data storage nodes. For each execution we measured the average throughput achieved when multiple concurrent clients perform the same operation on the Cumulus service. The clients are launched simultaneously on dedicated machines. Each client performs an upload and a download for a single file of 1 GB. We increased the number of concurrent clients from 1 to 200 and we measured the average throughput of each operation. The results are shown in Figure 2. As expected,



(a) Upload average throughput for 30 Cumulus servers. (b) Download average throughput for 30 Cumulus servers.

Fig. 2. Cumulus storage backend comparison under concurrent accesses

when each Cumulus server uses its own local disk for storage, the performance of the data transfers is better than when Cumulus is backed by a distributed file system. As the number of clients increases, the available network bandwidth is divided among the concurrent requests, resulting in lower average throughputs. As the number of clients increases to more than 100, the gap between the local disk backend and the distributed file systems is substantially reduced, due to the efficient data striping of the latter and the contention generated by the concurrent accesses when using the local disks. However, PVFS is not optimized for writing small blocks of data and its consistency mechanisms do not allow any client-side caching. As a result, PVFS cannot achieve its raw performance level when being used as a storage backend for the POSIX interface of Cumulus.

In contrast, the BlobSeer-based backend is specifically tuned to take full advantage of the storage system features. Despite the fact that the number of concurrent clients reaches 6 times the number of data storage nodes (which amounts to 30 nodes), the BlobSeer-backed system sustains an almost constant transfer rate for more than 60 clients and transferred data amounting to 180 GB. Thus, BlobSeer outperforms PVFS, maintaining a throughput approximatively 30% higher in the case of uploads and 60% higher for downloads.

5.3 Evaluating Cumulus as a Cloud Service for CM1

To study the impact of storing HPC application data in a Cloud service, we relied on Nimbus to provide a customizable IaaS Cloud environment on Grid'5000. We employed 64 nodes to deploy Nimbus. For each experiment, the CM1 application was executed in a Nimbus virtual cluster of quadcore VMs with 4 GB of RAM. Each VM of such a cluster was equipped with the S3-backed file system, to enable CM1 to execute in parallel on the virtual cluster nodes, and to directly store its output data into Cumulus, as shown in Figure 3(a). We used 50 nodes to deploy the replicated Cumulus servers on top of the three storage backends: the local file system of each Cumulus node, BlobSeer and PVFS. Both BlobSeer and PVFS employ 50 *data providers* and 10 *metadata nodes*, each of them being deployed on dedicated machines.

CM1 is representative for a wide class of applications that simulate the evolution of a phenomenon in time. Each simulation is associated with a 3D domain and a time

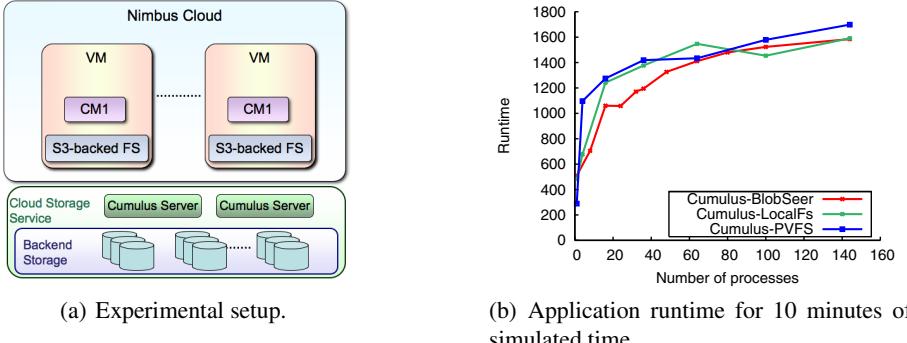


Fig. 3. Cloud storage evaluation for CM1 in a Nimbus Cloud environment

interval. A simulation consists in obtaining the values for a set of parameters for each point of the domain and for each time step. The initial domain is split among the processes and each of them is in charge of a particular subdomain. We used a 3D hurricane simulation described in [4]. The application was configured to use MPI, each process generating a set of output files for each time step.

Completion time when increasing the pressure on the storage system. For the first experiment, we executed the application for 10 minutes of simulated time, with a time step of 20 seconds. We increased the number of MPI processes, maintaining constant the size of the simulated subdomain for each of them. We generated 4 intermediate output files, each of them of 85 MB in size, amounting to a total of 340 MB generated by each process per run. We deployed 4 MPI processes on each virtual machine (one for each core) and increased the number of processes from 1 to 144. The total size of the data generated for these simulations increases from 340 MB to 50 GB. Figure 3(b) shows the simulation completion time when increasing the number of processes and the output data is stored into Cumulus.

The results show a very steep increase of the completion time when the number of processes is small. However, for more than 20 processes, the curve flattens for all three storage solutions. The measured runtime trend can be explained by the increasing size of the simulated domain, which leads to a larger time spent in communication among the MPI processes that need to exchange more subdomain border values. In addition, the size of the output data increases along with the number of processes. The sustained performance for a large number of processes thus suggests the application is able to scale despite storing output data into the Cumulus external repository. The results also indicate the BlobSeer and the PVFS backends for the Cumulus servers do not lead to a performance drop for the application that stores data into Cumulus, when compared against the local file system backend. Moreover, the BlobSeer-based Cumulus version slightly outperforms the PVFS Cumulus backend. Similarly to the Cumulus benchmarks in the previous section, this result confirms the higher throughput delivered by BlobSeer in this context.

Application speedup. This experiment aims at evaluating the speedup obtained by scaling out the number of application processes for the same initial problem. For this

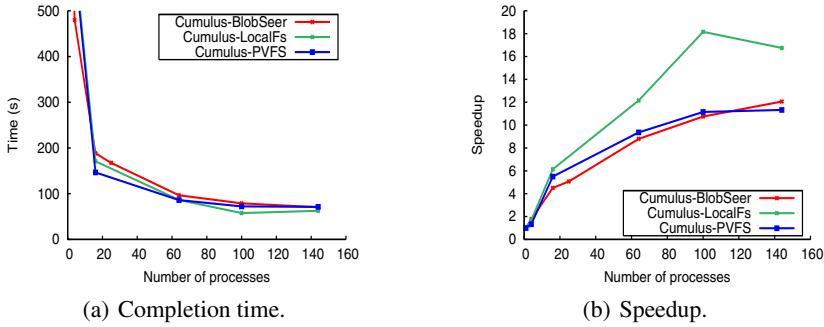


Fig. 4. Storage backend comparison for 30 minutes of simulated time with a 20 s time step.

evaluation, CM1 was executed for 30 minutes of simulation time, with a timestep of 20 seconds between consecutive computations. In contrast to the previous experiment, we increased the duration of the computation phase, so as to highlight the benefits of parallelizing the simulation on more processing nodes. For each point on the graph, Figure 4(a) depicts the time it takes the application to complete when the initial problem is divided among an increasing number of processes. Additionally, we have shown the corresponding speedup in Figure 4(b). The speedup for a specific number of processes is computed as the measured execution time of the application for a single process divided by the execution time when all the processes are employed.

As expected, as we decrease the size of the simulated spatial domain allocated to each process, the application completes its execution much faster. The drop in the execution time as we introduce new processes is a consequence of the smaller number of border values to be exchanged between them and the smaller size of the output data to be sent to the storage backend. The obtained performance of the three backends is similar, as most of the execution time accounts for computation. Furthermore, the two distributed backends achieve a comparable speedup. We however obtained a better speedup when using the local file system of each Cumulus server as a storage solution. This result is mainly due to the large execution time measured for the local file system in the case of only one process, when the all generated data had to be dumped on a single node's file system, whereas it was distributed among storage nodes for the two other backends.

6 Conclusions

In this paper we address the data management needs of tightly-coupled scientific applications executed in Cloud environments. In this context, we argue that Cloud storage services can meet some of the requirements of such applications and contribute to the adoption of the Cloud paradigm by the HPC community. The benefit of such an approach lies in the ability to provide globally-shared access to data generated by parallel processes and to enable simultaneous data analysis or visualization. We assess the performance of an open-source Cloud data service called Cumulus, backed by various storage solutions. Among them, we focused on BlobSeer, a data management system optimized for highly-concurrent accesses to large-scale distributed data. We proposed an interfacing mechanism that enables Cumulus to use BlobSeer as a storage backend

and to take advantage of its efficient data transfer operations while exposing an S3-compatible interface to the users. Furthermore, we evaluated Cumulus for CM1, a real-life tightly-coupled simulator for atmospheric phenomena. We conducted experiments on a private Nimbus Cloud deployed on the Grid'5000 infrastructure, showing that a data service such as Cumulus allows the application to scale both with the number of processes and with the size of the generated output. We also examined various backends for Cumulus, the obtained results suggesting the backend choice has an impact on the completion time of the executed application, along with providing additional benefits for higher-level data analysis tools, such as data availability or standard interfaces.

As future work, we plan to perform more in-depth evaluations for various workloads and access patterns, as well as to study the advantages of leveraging Cumulus for online visualization of the generated simulation results.

Acknowledgments. This work has been supported by the ANR MapReduce grant (ANR-10-SEGI-001). The experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies.

References

1. Amazon Elastic Block Store (EBS), <http://aws.amazon.com/ebs/>
2. Amazon Elastic Compute Cloud (EC2), <http://aws.amazon.com/ec2/>
3. Bresnahan, J., Keahey, K., LaBissoniere, D., et al.: Cumulus: an open source storage cloud for science. In: ScienceCloud 2011, pp. 25–32. ACM, New York (2011)
4. Bryan, G.H., Rotunno, R.: Evaluation of an analytical model for the maximum intensity of tropical cyclones. *Journal of the Atmospheric Sciences* 66(10), 3042–3060 (2009)
5. Carlyle, A.G., Harrell, S.L., Smith, P.M.: Cost-effective HPC: The community or the Cloud? In: CloudCom 2010, pp. 169–176 (December 2010)
6. Cloud Model 1, <http://www.mmm.ucar.edu/people/bryan/cm1/>
7. Ekanayake, J., Fox, G.: High Performance Parallel Computing with Clouds and Cloud Technologies. In: Avresky, D.R., Diaz, M., Bode, A., Ciciani, B., Dekel, E. (eds.) *Cloudcomp 2009*. LNICST, vol. 34, pp. 20–38. Springer, Heidelberg (2010)
8. El-Khamra, Y., Hyunjoo, K., Shantenu, J., et al.: Exploring the performance fluctuations of HPC workloads on clouds. In: CloudCom, pp. 383–387 (December 2010)
9. File System in UserspacE (FUSE), <http://fuse.sourceforge.net>
10. Gropp, W., Lusk, E., et al.: High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing* 22(6), 789–828 (1996)
11. He, Q., Zhou, S., Kobler, B., et al.: Case study for running HPC applications in public clouds. In: HPDC 2010, USA, pp. 395–401 (2010)
12. Jégou, Y., Lantéri, S., Leduc, J., et al.: Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *Intl. J. of HPC Applications* 20(4), 481–494 (2006)
13. Keahey, K., Freeman, T.: Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In: Cloud Computing and Its Applications (CCA), USA (2008)
14. Ligon, W.B., Ross, R.B.: Implementation and performance of a parallel file system for high performance distributed applications. In: HPDC 1996, pp. 471–480. IEEE Computer Society, Washington, DC (1996)

15. Nicolae, B., Antoniu, G., Bougé, L., et al.: BlobSeer: Next generation data management for large scale infrastructures. *J. Parallel and Distrib. Comput.* 71(2), 168–184 (2011)
16. Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In: Avresky, D.R., Diaz, M., Bode, A., Ciciani, B., Dekel, E. (eds.) *Cloudcomp 2009*. LNICST, vol. 34, pp. 115–131. Springer, Heidelberg (2010)
17. Amazon Simple Storage Service (S3), <http://aws.amazon.com/s3/>
18. Schmuck, F.B., Haskin, R.L.: GPFS: A shared-disk file system for large computing clusters. In: Conf. on File and Storage Technologies, FAST, pp. 231–244. USENIX (2002)
19. VisIt, <https://wci.llnl.gov/codes/visit/>
20. Younge, A.J., Henschel, R., Brown, J.T., et al.: Analysis of virtualization technologies for high performance computing environments. In: CLOUD, pp. 9–16 (July 2011)
21. Zhai, Y., Liu, M., Zhai, J., et al.: Cloud versus in-house cluster: Evaluating amazon cluster compute instances for running mpi applications. In: 2011 Intl. Conf. for HPC, Networking, Storage and Analysis, SC, pp. 1–10 (November 2011)

Targeting Distributed Systems in FastFlow*

Marco Aldinucci¹, Sonia Campa², Marco Danelutto²
Peter Kilpatrick³, and Massimo Torquati²

¹ Computer Science Department, University of Torino, Italy

² Computer Science Department, University of Pisa, Italy

³ Computer Science Department, Queen's University Belfast, UK

Abstract. **FastFlow** is a structured parallel programming framework targeting shared memory multi-core architectures. In this paper we introduce a **FastFlow** extension aimed at supporting also a network of multi-core workstations. The extension supports the execution of **FastFlow** programs by coordinating—in a structured way—the fine grain parallel activities running on a single workstation. We discuss the design and the implementation of this extension presenting preliminary experimental results validating it on state-of-the-art networked multi-core nodes.

Keywords: structured parallel programming, multi-core, fine grain.

1 Introduction

In a scenario with more and more cores per socket available to the application programmer it is becoming increasingly urgent to provide programmers with effective parallel programming tools. Programming tools and frameworks are needed to efficiently target the architectures hosting inter networked, possibly heterogeneous, multi-core devices, which appear to be “the” reference architecture ferrying programmers from the mainly sequential to mainly parallel programming era [1]. The urgency is even more crucial given that both grids and clouds provide application programmers with the possibility to reserve collections of multi-cores to support parallel applications eventually presented as, or orchestrated by, web services.

Shared memory multi-core and clusters/networks of processing elements, however, require quite different techniques and tools to support efficient parallelism exploitation. The *de facto* standard tools in the two cases are OpenMP [2] and MPI [3] used either alone or in conjunction. Despite being very efficient on some classes of applications, OpenMP and MPI share a common set of problems: poor separation of concerns between application and system aspects, a rather low level of abstraction presented to the application programmer and poor support for really fine grained applications are all considerations hindering easy use of MPI and OpenMP. Actually, it is not even clear yet if the mixed MPI/OpenMP programming model always offers the most effective mechanisms for programming clusters of SMP systems [4].

* This work has been partially supported by ParaPhrase (www.paraphrase-ict.eu)

The algorithmic skeleton community has proposed various programming frameworks aimed at providing the application programmer with very high level abstractions completely encapsulating parallelism exploitation patterns and solving most of the problems mentioned above [5,6]. Initial skeleton based programming frameworks targeted only cluster/network of workstations. More recently, some of the existing frameworks have been extended in such a way clusters of multi-core may also be exploited. SkeTo [7] provides data parallel skeletons as proper C++ abstractions. It has been recently extended to target multi-core clusters using a two-step dynamic task scheduling strategy, which enables balancing of the load both between nodes of the cluster and cores of the single node [8]. The Muesli programming framework is provided as a C++ library offering both data and stream parallel skeletons [9]. The original version of Muesli generated code for MPI platforms. Recently, Muesli has been extended in such a way that both multi-core architectures and distributed multi-core workstations may be targeted by generating OpenMP code in addition to the MPI code [10].

The contribution of this paper is twofold. First, we discuss an extension of **FastFlow** targeting clusters of multi-core workstations. The extended version supports a two tier parallel model with a lower tier exploiting fine grain parallelism on single multi/many core workstation and the upper layer supporting structured coordination—across internetworked workstations—of medium/coarse grain parallel activities. Second, we present experimental results validating the proposed approach and showing: i) how real applications may be structured using the proposed model, ii) how different interconnection networks may be seamlessly exploited depending on the communication bandwidth required by the application, and iii) how our programming framework is able to fully exploit state-of-the-art architectures.

2 The FastFlow Programming Framework

FastFlow¹ is a structured parallel programming environment implemented in C++ on top of the Pthreads library and targeting shared memory multi-core. **FastFlow** provides programmers with predefined and customizable task farm and pipeline parallel patterns. It has been initially designed and implemented to be very efficient in the execution of fine grain parallel applications [11].

The **FastFlow** design is layered (see Fig. 1). The lower layer implements a very efficient, lock free and wait free single producer, single consumer queue [12]. On top of this mechanism, the second layer provides single producer multiple consumer and multiple producer single consumer queues. Finally, the third layer provides the *farm* and *pipeline* parallel patterns as C++ classes [13].

The key concept in the implementation of **FastFlow** is the `ff_node` class. It is used to encapsulate sequential portions of code implementing functions as well as higher level parallel patterns such as pipelines and farms. The `ff_node` class structure is outlined in Fig. 1. Each `ff_node` will be used to run a concurrent activity in a thread, and it has associated two (shared memory) message queues:

¹ Project site <http://sourceforge.net/projects/mc-fastflow/>

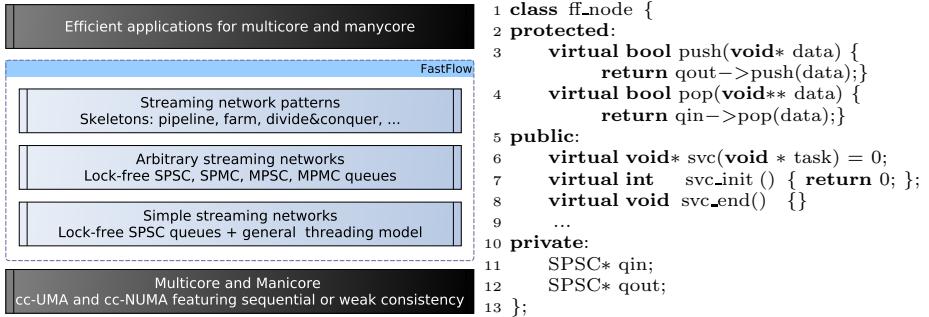


Fig. 1. Left: Layered **FastFlow** design. Right: **FastFlow**'s `ff_node` class schema.

one used to receive input data (pointers) to be processed and one to deliver the (pointers to) computed results. The `svc` method encapsulates the computation to be performed on each input datum to obtain the output result. `svc_init` and `svc_end` methods are executed when the application is started and before it is terminated. The three methods constitute the only code the programmer has to provide to instantiate an `ff_node`.

The predefined patterns provided by **FastFlow** may be customized in different ways. For example default patterns may be arbitrarily nested and so we can have pipelines with farm stages and vice versa. Using the customization features, different patterns may be implemented in terms of the pipe and farm building blocks, such as divide&conquer, map and MISD² pattern.

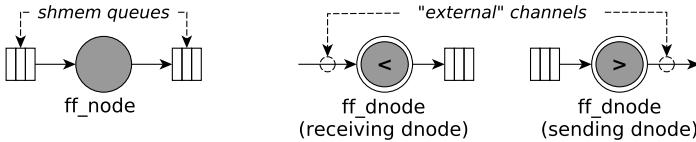
FastFlow is being currently extended to support also data parallel patterns and to offload data parallel computation to GPUs, where available.

3 From Single to Multiple Nodes

FastFlow was originally designed to target efficiently shared-cache multi-core platforms and only those platforms. The **FastFlow** stream semantics guarantees correct sequencing of activation of the concurrent activities modeled through `ff_nodes` and connected through streams. The stream implementation ensures *pure data flow* semantics.

In order to scale to thousands of cores in terms of computational power, or to be able to use huge amounts of memory, the only viable solution is to exploit more and more multi-core workstations together. To this end, the streaming network model provided by **FastFlow** to describe concurrent activities turns out to be suitable for use also in loosely-coupled systems such as clusters, cloud and grids. The idea has been therefore to extend the **FastFlow** framework to provide the user with a two-tier programming model:

² Different computations on the same input, providing a vector result (each position hosting the result of one computation).



```

1 template<typename CommImpl>
2 class ff_dnode: public ff_node {
3 protected:
4     bool
5     push(void* data){... return com.put(data);}
6     bool
7     pop(void** data){... return com.get(data);}
8 public:
9     int
10    init(std::string& name, std::string& address,
11          int peers, CommImpl::TransportImpl* transp,
12          bool p, int nodeId, dnode_cbk_t cbk=0) {
13      ... return com.init(address, nodeId);
14  }
15     // serialization/deserialization methods
16     // used by the sender dnode
17     virtual void
18     prepare(svector<iovec>& v, void* ptr);
19     // used by the receiver dnode
20     virtual void
21     prepare(svector<msg_t*>*& v, size_t len);
22     virtual void
23     unmarshalling(svector<msg_t*>* v[],
24                    int vlen, void*& task);
25 private:
26     CommImpl com;
27 };
28 };

```

Fig. 2. Top)FastFlow’s node vs dnode(s). Bottom) FastFlow’s dnode class schema.

- at a *lower tier*, a shared-memory implementation of skeletons inside a single multi-core workstation is supported;
- at an *upper tier*, structured coordination among a set of distributed nodes executing the lower tier computations is supported, by providing all the mechanisms needed to implement the low tier skeletons in the distributed multi-core scenario.

More specifically, at the lower tier the user designs a typical **FastFlow** skeleton graph, employing stream parallelism and the shared memory skeletons offered by the original **FastFlow** framework. Parallel patterns implement structured synchronization among concurrent entities (graph nodes) via shared memory pointers passed in a consumer-producer fashion. The **FastFlow** run time support takes care of all the required synchronization relating to communication among the different nodes resulting from the compilation of the high level **FastFlow** pattern(s) used in an application. At this level, the entire **FastFlow** graph describing the application is implemented using non-blocking concurrent threads inside a single process abstraction. Then, multiple lower tier **FastFlow** graphs can be connected together using the mechanisms of the second tier, that is, using a suitable communication pattern which implements a network channel (i.e. point-to-point, broadcast, scatter, etc.). At this level, the programming model exposed to the programmer can be either SPMD or MPMD.

In order to send and receive tasks from and to other **FastFlow** graphs, the edge-nodes of the **FastFlow** application have to be defined as **ff_dnode**.

A **ff_dnode** is actually a **ff_node** with an extra communication channel (henceforth *external channel*) which connects the edge-node of the graph with one or more edge nodes of other **FastFlow** application graphs running on the same or on a different host. At the second tier no memory is shared among

processes and so all iterations have to be implemented using explicit communications, which are the responsibility of the **FastFlow** run-time support and are completely transparent to the user due to the careful design of the **ff_dnode**.

As shown at the top of Fig. 2, a **FastFlow** node is a concurrent activity with an input and an output channel which are implemented using concurrent queues, whereas a **ff_dnode** is a concurrent activity where one of the 2 channels (either the input or the output one) is an external channel. A **ff_dnode** cannot have both external input and output channels at the same time since the minimal pure **FastFlow** application is composed of at least 2 nodes (a pipeline of two sequential nodes or a farm with an Emitter node and a sequential worker node). The interface of the **ff_dnode** class is sketched at the bottom of Fig. 2. The template parameter (**CommImpl**) represents the type of the communication pattern that the programmer wishes to use to connect different **ff_dnodes**. The **ff_dnode::init** method has to be called in order to initialize the external channel and it is typically called within the **ff_node::svc_init** method in order to perform parallel initialization among multiple **ff_dnode**³. If the **ff_dnode::init** method is not called the dnode behaves like a pure **FastFlow** node. The **ff_dnode** class overwrites the *pop* and *push* methods of the **ff_node** class so that they work on external input and output channels if activated by the **ff_dnode::init** method.

ZeroMQ as External Transport Layer. ZeroMQ is an LGPL open-source communication library [14]. It provides the user with a socket layer that carries whole messages across various transports: inter-thread communications, inter-process communications, TCP/IP and multicast sockets. ZeroMQ offers an asynchronous communication model, which allows construction of complex asynchronous message-passing networks very quickly and with reasonable performance. The message API offers the possibility to perform zero-copy sends and non-blocking calls to the socket layer.

In **FastFlow** we used ZeroMQ as the external transport for the **ff_dnode** concurrent entity. In particular, we built on top of ZeroMQ all the communication patterns using the **DEALER** and the **ROUTER** sockets offered by the library. The **ROUTER** socket allows routing of messages to specific connections provided that the peer identifier is known; the **DEALER** socket instead can be used for fair-queuing on input and for performing load-balancing on output toward a pool of connections. Within **FastFlow** we do not use the load-balancing feature; instead we use it to connect a **ff_dnode** to a **ROUTER** socket of another **ff_dnode**. The ease-of-use of ZeroMQ was the factor for choosing it for the implementation of the distributed transport layer.

Communication Patterns Among ff_dnode(s). A dnode's external channel can be specialized to provide different patterns of communication. The set of communication collectives, allows exchange of messages among a set of distributed nodes using well-known predefined patterns. The semantics of each communication pattern currently implemented, may be summarized as follows:

³ The **svc_init** method is called once when the node thread has already started.

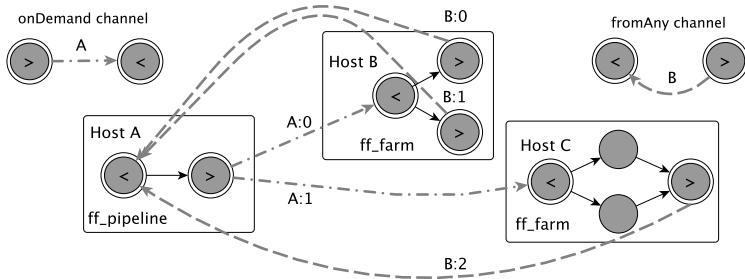


Fig. 3. An example application schema that represents 3 **FastFlow** applications connected through one *onDemand* and one *fromAny* communication pattern

<i>unicast</i>	unidirectional point-to-point communication between two peers
<i>broadcast</i>	sends the same input data to all connected peers
<i>scatter</i>	sends different parts of the input data (typically partitions) to all connected peers
<i>onDemand</i>	the input data is sent to one of the connected peers, the choice of which is taken at run-time on the basis of the actual work-load (typically it is implemented using a request-reply protocol)
<i>fromAll</i>	(also known as all-gather) collects different parts of the data from all connected peers combining them in a single data item
<i>fromAny</i>	collects one data item from one of the connected peers

Each communication pattern has a unique identifier (the channel-name). Each *ff_dnode* has an identifier (not necessarily unique) for the external channel in the range [0..*num-peers*] so that the tag “channel-name:*dnode-id*” is used as a unique identity for the peer connection. Fig. 3 sketches a possible interaction among different **FastFlow** applications (one Pipeline on the HostA, and two different Farm skeletons on the HostB and on the HostC) connected via an *onDemand* communication pattern, whose channel name is “A”, used to feed the 2 farms with input tasks, and a *fromAny* pattern, whose channel name is “B”, used to collect back to the HostC the results in a non-deterministic order.

Marshalling and Unmarshalling. The *ff_dnode* class provides the user with suitable methods that can be overwritten in order to manage zero-copy marshalling and unmarshalling of data. A *prepare* method allows serialization of non-adjacent data on the sending side, while a *prepare* and a *unmarshalling* methods at the receiving side allow deserialization of the received data in properly cast-able buffers. We do not give more details of this due to space limitations.

4 Experiments

All experiments were run on 2 workstations running Linux x86_64 (named HostA and HostB) equipped with 2 Intel Sandy Bridge Xeon E5-2650 CPUs @2.0GHz. Each CPU has 8 cores double context, 20MB L3 shared cache and 32 GBytes

size	8B	8KB	1MB
	Latency (μ s)	Bandwidth (Gb/s)	
Eth	69	0.93	0.95
Inf	27	5.1	14.7

(Latency measured using a torus of 2 FastFlow 2-stage pipelines whose edge-nodes are connected with a unicast channel)

Fig. 4. Latency and bandwidth of the *unicast* channel for both 1Gb Ethernet and IP over IB Infiniband networks

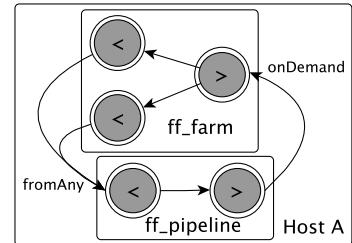


Fig. 5. Parallel schema of the benchmark test when executed on a single host

of memory. The two workstations are connected both with a Gigabit Ethernet and an Infiniband Connectx-3 card (40 Gb/s ports) using a direct cable. Since ZeroMQ does not have any native support for Infiniband, we used the IP over IB driver. We used two workstations to be able to evaluate precisely the latency and throughput of the FastFlow extension.

We first measured the raw performance of the new *unicast* channel (see Fig. 4). The maximum bandwidth obtained running the *iperf* benchmark tool (v.2.0.5)⁴ over the Infiniband network using TCP/IP is 4.8 Gb/s and 17.6 Gb/s for 8KB and 1MB messages respectively. This means that the *unicast* channel does not introduce any significant overhead and is able to saturate almost all the available network bandwidth.

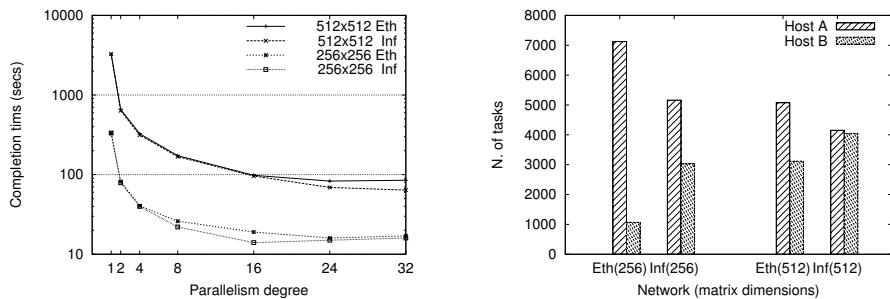


Fig. 6. Benchmark test executed on the two hosts using a stream of 256×256 and 512×512 matrices with the 1Gb Ethernet (Eth) and the Infiniband (Inf) networks. Left: Completion time in seconds. Right: Number of tasks computed in the 2 hosts.

⁴ <http://sourceforge.net/projects/iperf>

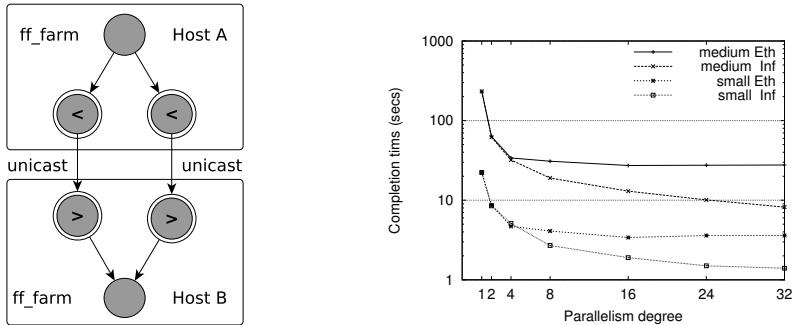


Fig. 7. Image filter application. Left: Application schema. Right: Completion time for small and medium sized images on the 2 networks.

Table 1. Left: Benchmark test (see Fig. 5 for the dFF-1 schema). Right: Image filter application (see Fig. 7 left for dFF-2 schema). Completion time in seconds obtained running different implementations.

mat. size	FF	dFF-1	dFF-2-Eth	dFF-2-Inf	img. size	FF	dFF-2-Eth	dFF-2-Inf
256×256	24.4	18.9	16	14	small	2.4	3.4	1.4
512×512	203	158	83	64	medium	19.2	27.2	8.2

In order to test the impact of computation grain and communication bandwidth and latency, we used a synthetic application computing the square of a stream of input matrices. We ran two experiments: one using 256×256 and the other using 512×512 matrices of doubles. We set the stream length to 8192 matrices for a total of 4GB and 16GB of input data, respectively. The parallelization schema adopted is very similar to that sketched in Fig. 3, where 2 FastFlow farms (without the collector) and one 2-stage pipeline are used. The first stage of the pipeline generates the input matrices and then waits for the results from the workers of the two farms. The second stage schedules (using the *ondemand* communication pattern) the matrices to the two farms. The sequential time to compute the square of all 8192 matrices is 333s for the 256×256 and 3260s for 512×512 matrices. As shown in Table 1 (left), the distributed version of the application (dFF-1) running the FastFlow pipeline and just one FastFlow farm on the same HostA (the parallel schema is sketched in Fig. 5), obtains $17.6 \times$ speedup for 256×256 matrices and $20 \times$ speedup for 512×512 matrices using 16 worker threads. The better result obtained with respect to the shared memory FastFlow version of the same benchmark (FF in the table), is justified by considering that pure FastFlow applications use only non-blocking threads to implement skeletons in order to minimize overheads when executing fine-grained tasks. However, when running medium to coarse-grained computations, non-blocking execution may increase contention and memory pressure not allowing increase of the number of threads to a level greater than the number of physical

cores, in order to exploit multi-context platforms. In the distributed version, the *get* operations on the external channel are blocking, thus contention is reduced resulting in better performance. The distributed **FastFlow** version running on a single host is able to saturate all the available memory bandwidth of the tested platform (51.2 GB/s), reaching a maximum memory throughput of 51.8 GB/s. Figure 6 (left) sketches the performance obtained when 2 farms are used (**dFF-2** in Table 1 right), one running on HostA (together with the pipeline) and one on HostB. In this case the speedup is $23\times$ for 256×256 matrix and $\sim 51\times$ for 512×512 when the Infiniband network is considered. The super-linear speedup obtained is most likely due to the better L3 cache utilization on both hosts which provides the farm skeletons with more memory bandwidth. Figure 6 (right) shows the number of tasks computed by each farm. When the Ethernet network is used, many more tasks are scheduled toward HostA (the one where the pipeline is mapped) because of the lower latency of the communication. As expected, the smaller the granularity, the larger the number of tasks computed on HostA for both networks.

Finally, we ran another test using a simple streaming application: two image filters (blur and emboss) are applied to a stream of input GIF images. The stream can be of any length and images of any size. In our test we fixed the number of images to 256, considering small size images (256KB) and coarser size images (1.7MB) as two separate test cases. The application uses the ImageMagick library⁵ to manipulate the images and to apply the filters. The times reported in this paper do not consider the time spent to read and write the images from/to disk. The sequential time to compute all images is 27.5s and 232s for small and medium images, respectively. In Table 1 (right) is reported the minimum completion time obtained for the tested versions. The pure shared memory **FastFlow** version (**FF**) obtains a $11\times$ speedup and $12\times$ speedup for the two sizes. The distributed version uses a pipeline of 2 farms running on the 2 hosts (**dFF-2**). The first farm computes the blur and the second computes the emboss filter. The workers of the 2 farms are directly connected using a *unicast* channel (see the parallel schema in Fig. 7 right side). The left side of Fig. 7 shows the completion time of the distributed version. The maximum speedup obtained is $8\times$ and $8.5\times$ speedup when using the Gigabit Ethernet, and $19.6\times$ and $28.3\times$ when using the Infiniband network, which represent fairly good results on a 2×16 core cluster.

5 Conclusions

We have proposed an extension of the **FastFlow** programming framework suitable for targeting clusters of multi-core workstations. Using a small set of applications, we have demonstrated that the extended **FastFlow** succeeds in exploiting resources in a cluster of workstations with different interconnection networks. We are currently working to implement the higher tier “algorithmic skeletons” in such a way that application programmers may seamlessly implement extended **FastFlow** applications much in the same way that they use to implement “single

⁵ <http://www.imagemagick.org>

multi-core” applications with the original framework. The whole activity—along with the activities aimed at supporting GPUs within **FastFlow** [15]—is intended to provide suitable means to implement the computing model designed within ParaPhrase, an FP7 STREP project whose goal is to use parallel design patterns and algorithmic skeletons to program heterogeneous—multi-core plus GPU—collections of processing elements.

References

1. Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., Yelick, K.: A view of the parallel computing landscape. *Comm. of the ACM* 52(10), 56–67 (2009)
2. Park, I., Voss, M.J., Kim, S.W., Eigenmann, R.: Parallel programming environment for OpenMP. *Scientific Programming* 9, 143–161 (2001)
3. Pacheco, P.S.: Parallel programming with MPI. Morgan Kaufmann Publishers Inc., San Francisco (1996)
4. Cappello, F., Etiemble, D.: Mpi versus mpi+openmp on ibm sp for the nas benchmarks. In: Proc. of the 2000 ACM/IEEE Conference on Supercomputing (CDROM), Supercomputing 2000. IEEE Computer Society (2000)
5. Cole, M.: Bringing skeletons out of the closet: A pragmatic manifesto for skeletal parallel programming. *Parallel Computing* 30(3), 389–406 (2004)
6. González-Vélez, H., Leyton, M.: A survey of algorithmic skeleton frameworks: High-level structured parallel programming enablers. *Software: Practice and Experience* 40(12), 1135–1160 (2010)
7. Matsuzaki, K., Iwasaki, H., Emoto, K., Hu, Z.: A library of constructive skeletons for sequential style of parallel programming. In: Proc. of the 1st Inter. Conference on Scalable Information Systems, InfoScale 2006. ACM, New York (2006)
8. Karasawa, Y., Iwasaki, H.: A parallel skeleton library for multi-core clusters. In: Proceedings of the 2009 International Conference on Parallel Processing, ICPP 2009, pp. 84–91. IEEE Computer Society, Washington, DC (2009)
9. Ciechanowicz, P., Poldner, M., Kuchen, H.: The Munster skeleton library Muesli – a comprehensive overview. In: ERCIS Working paper. Number 7 (2009)
10. Ciechanowicz, P., Kuchen, H.: Enhancing muesli’s data parallel skeletons for multi-core computer architectures. In: Proceedings of the 2010 IEEE 12th International Conference on High Performance Computing and Communications, HPCC 2010, pp. 108–113. IEEE Computer Society, Washington, DC (2010)
11. Aldinucci, M., Danelutto, M., Kilpatrick, P., Torquati, M.: Fastflow: high-level and efficient streaming on multi-core. In: Programming Multi-core and Many-core Computing Systems. Parallel and Distributed Computing. Wiley (2012)
12. Aldinucci, M., Danelutto, M., Kilpatrick, P., Meneghin, M., Torquati, M.: An Efficient Unbounded Lock-Free Queue for Multi-core Systems. In: Kaklamanis, C., Papathodorou, T., Spirakis, P.G. (eds.) Euro-Par 2012. LNCS, vol. 7484, pp. 662–673. Springer, Heidelberg (2012)
13. Aldinucci, M., Danelutto, M., Torquati, M.: Fastflow tutorial. Technical Report TR-12-04, Università di Pisa, Dipartimento di Informatica, Italy (March 2012)
14. ZeroMQ: website (2012), <http://www.zeromq.org/>
15. Aldinucci, M., Danelutto, M., Kilpatrick, P., Torquati, M.: Targeting heterogeneous architectures via macro data flow. *Parallel Processing Letters* 22(2) (June 2012)

Throughput Optimization for Pipeline Workflow Scheduling with Setup Times

Anne Benoit¹, Mathias Coqblin², Jean-Marc Nicod², Laurent Philippe²,
and Veronika Rehn-Sonigo²

¹ LIP, ENS Lyon and Institut Universitaire de France

² FEMTO-ST Institute, CNRS/UFC/ENSMM/UTBM, Besançon

Abstract. We tackle pipeline workflow applications that are executed on a distributed platform with setup times. Several computation stages are interconnected as a linear application graph, and each stage holds a buffer of limited size where intermediate results are stored and a processor setup time occurs when passing from one stage to another. In this paper, we focus on interval mappings (consecutive stages mapped on a same processor), and the objective is the throughput optimization. Even when neglecting setup times, the problem is NP-hard on heterogeneous platforms and we therefore restrict to homogeneous resources. We provide an optimal algorithm for constellations with identical buffer capacities. When buffer sizes are not fixed, we deal with the problem of allocating the buffers in shared memory and present a $b/(b+1)$ -approximation algorithm.

1 Introduction

In this paper, we consider pipeline workflow applications mapped on a distributed platform such as a grid. This kind of applications is used to process large data sets or data that are continuously produced by some source and produce some final results. The first stage of the pipeline is applied to an initial data to produce an intermediate result that is then sent to the next stage of the pipeline and so on until the final result is computed. Examples of such applications include image set processing where the different stages may be filters, encoders, image comparison or merging and video capture processing and distribution where codecs must be applied on the video flow before being delivered to some device. In this context, a first scheduling problem is to map the pipeline stages on the processors. Subhlok and Vondran [13,14] show that there exists an optimal interval mapping for a given pipeline and a given platform when communications and processors are homogeneous. An interval mapping is defined as a mapping where only consecutive pipeline stages are mapped on the same processor. However, the cost of switching between stages of the application on one processor is not taken into account. When a new data set arrives on the processor, the first local stage starts to process it as soon as the previous data set is output. Then this data set moves from stage to stage until the last local stage, and it is sent to the processor in charge of the following stage. So,

at each step of the execution, we switch from one stage to the next one. As a result, if the cost of switching cannot be neglected, several setup times must be added to the processing cost. Benoit and Robert [4] prove that the basic interval mapping problem is NP-hard as soon as communications or computations are heterogeneous, even without setup times. For this reason, we restrict this work to homogeneous platforms.

The problem of reconfiguration that requires a setup time has been widely studied, and covers a lot of domains (see survey [1]). For instance, Zhang et al. [15] address the problem of wafer-handling robot calibration in semiconductor factories. They propose a low-cost solution to reduce the robot end effector tolerance requirements and thus the calibration times. A solution based on ant colony optimization is proposed in [9,10] to reduce the setup costs in batch processing of different recipes of semiconductors. In the scope of micro-factories, due to the cost of design and production of micro-assembly cells, micro-assembly cells are being designed with a modular architecture that can perform various tasks, at the cost of a reconfiguration time between them [8]. In the domain of pure computing, setup times may appear when there is a need to swap resources, or to load a different program in memory, e.g., to change the compiler in use [2]. Some authors have also shown interest in using buffers to stock temporary results after each stage of the pipeline, in order to reduce the amount of performed setups. Bryan and Norman [7] consider a flowshop wherein a job consists of m stages mapped on m processors, and a processor must be reconfigured after each job to process the next one (in their example, the clean-out of a reactor in a chemical processing facility). They acknowledge that the problem of sequence-dependent setup times in which a setup time depends on the previous stage and the next one is NP-hard and they propose several heuristics. Luh et al. [11] study scheduling problems in the manufacturing of gas insulated switchgears. The problems involve significant setup times, strict local buffer capacities, and few possible processing routes.

However, most of those researches consider that the number of processors is large enough to map each stage on only one processor (one-to-one mapping) and no reconfiguration is required before the next batch. Note that the one-to-one mapping problem can be solved in polynomial time provided that communications are homogeneous [4]. In our approach, we consider that the number of stages is greater than the number of processors. We therefore focus on interval mappings, where several consecutive stages are mapped onto the same processor.

The difficulty of the mapping problem is twofold. First, as in classical interval mapping one has to decide how to cut the different stages of the pipeline workflow into intervals, hence which stages are mapped onto the same processor. Second, the schedule inside a processor has to be fixed. Switching continuously between stages may lead to a drop in performance (due to the setup times), whereas buffering the data and defining a schedule for the processing of stages may limit the number of setups. Hence buffers are introduced to store intermediate results. This makes it possible to perform one stage several times before switching to the next one.

Starting from the interval mapping results, we tackle in this paper the problem of optimizing the cost of switching between stages mapped on the same processor depending on the buffer sizes. In a first step, we consider the single-processor scheduling problem where a single processor has to process several consecutive and dependent pipeline stages. Each stage is associated with a buffer. Usually, these buffers are limited by the available memory in the system and the buffer size hence influences the possible schedules as it limits the number of repetitions. Several other parameters are also taken into account as the duration of each stage's setup, the homogeneity or heterogeneity of buffers, and the available memory. Once the single-scheduling problem has been dealt with, we study in a second step the overall execution of the pipeline (in terms of throughput). Because of buffer utilization, data is treated and forwarded in batches, which leads to a data flow in waves. This particular behavior has to be taken into account in the solution.

We formally define the optimization problem in Section 2. The main contributions follow: (i) we provide optimal algorithms when buffers are of fixed size (Section 3); and (ii) we discuss how to allocate memory to buffers on a single processor in Section 4, both from a theoretical perspective (optimal algorithm in some cases), and from a practical point of view ($b/(b+1)$ -approximation algorithm). Finally, we conclude and discuss future work in Section 5.

2 Framework

The application is a linear workflow application, or pipeline. It continuously processes a large amount of consecutive data sets. Formally, a pipeline is expressed as a set S of n stages: $S = \{S_1, \dots, S_n\}$. Each data set is fed into the pipeline and traverses the pipeline from one stage to another until the entire pipeline is passed. A stage S_i receives a task of size δ_i from the previous stage, treats the data set which takes a number of w_i computations, and outputs data of size δ_{i+1} . The output data of stage S_i is the input data of the next stage S_{i+1} .

The target platform is a set P of p homogeneous processors $P = \{P_1, \dots, P_p\}$ fully interconnected as a clique. Each processor P_u has a processing speed (or velocity) v , expressed in instructions per time unit, and a memory of size M . It takes X/v time units for P_u to execute X floating point operations. Each processor P_u is interconnected with a processor P_v via a bidirectional communication link of bandwidth β (expressed in input size units per time unit). We work with a linear cost model for communications, so it takes X/β time units to send or receive a message of size X from processor P_u to processor P_v . Furthermore communications are based on the bi-directional one-port model [5,6], where a given processor can send and receive at the same time, but for both directions can only support one message at a time. Distinct processor pairs can however communicate in parallel. Communications are non-blocking, i.e., a sender does not have to wait for its message to be received as it is stored in a buffer, and the communications can be covered by the processing times provided that a processor has enough data to process.

Each processor can process data sets from any stage. However to switch from an execution stage S_i to the next stage S_j , the processor P_u has to be recon-

figured. This induces setup times, denoted as st . The level of heterogeneity in setup times leads to different models: *uniform setup times* (st), where all setup times are fixed to the same value, *sequence-independent setup times* (st_i), where the setup time only depends on the next stage S_i to which the processor is re-configured, and *sequence-dependent setup times* ($st_{i,j}$) that depend on both the current stage S_i and the next stage S_j . The problem with sequence-dependent setup times requires to look for the best setup order in a schedule to minimize the impact of setup times. This has already been proven to be NP-hard, and can be modeled as a Traveling Salesman Problem (TSP) [12]. Hence we will not study this problem in this paper, and we focus on st and st_i instead.

To execute a pipeline on a given platform, each processor is assigned an interval of consecutive stages. Hence, we search for a partition of $[1..n]$ into $m \leq p$ intervals $K_k = [I_k, J_k]$ such that $I_k \leq J_k$ for $1 \leq k \leq m$, $I_1 = 1$, $I_{k+1} = J_k + 1$ for $1 \leq k \leq m - 1$ and $J_m = n$. Interval K_k is mapped onto a processor P_u . Once the mapping is fixed, the processor internal schedule has to be decided, since it influences the global execution time. Each processor is indeed able to perform sequentially its allocated stages. However, setup times are added each time a processor switches from one stage to another. To reduce setup times a processor may process several consecutive data sets for a same stage. The intermediate results are stored in buffers, and each stage S_i mapped on P_u has an input buffer B_i of size $m_{i,u}$.

The sizes of these input buffers depend on the memory size M available on P_u and on the number of allocated stages, as well as on the input data sizes. The capacity $b_{i,u}$ of buffer B_i is the number of input data sets that the buffer is able to store within the allocated memory $m_{i,u}$. Hence, a processor is able to process data sets for a stage S_i as long as B_i is not empty, and B_{i+1} is not full. Actually if S_i is the last stage of the interval mapped on P_u , we allocate an output buffer BO_u of size mo_u with a capacity bo_u .

The objective function is to maximize the throughput ρ of the application, $\rho = \frac{1}{\mathcal{P}}$, where \mathcal{P} is the average period of time between the output of two consecutive data sets. Therefore, we aim at minimizing the period of the application. Since our framework model allows us to cover communication time by computation time, \mathcal{P} is formally defined by: $\mathcal{P} = \max_u \left(\max (in(u), cpu(u), out(u)) \right)$, where $in(u)$, $cpu(u)$, $out(u)$ are respectively the mean time to input, process and output one data set onto $P_u \in P$. In the next two sections, we explicitly evaluate the application period depending on fixed or variable buffer sizes.

3 Fixed Buffer Sizes

In this section, we deal with the scheduling problem with fixed buffer sizes for both single and multiple processors. We consider that buffers that are allocated on the same processor P_u are homogeneous, i.e., they have the same capacity b_u .

Single Processor Scheduling ($b_i = b$). With a single processor, the mapping is known, since stages S_1 to S_n form a single interval. We propose a polynomial time greedy algorithm to solve the problem of single processor scheduling and

prove its optimality. The idea is to maximize the number of data sets that are processed for a stage between each setup. This is done by selecting a stage for which the input buffer is full and the output buffer is empty, so that we can compute exactly b data sets, where b is the number of data sets that fits in each buffer. Therefore, we compute b data sets for stage S_1 , hence filling the input buffer of S_2 , and then perform a setup so that we can compute b data sets for stage S_2 , and so on, until these b data sets exit the pipeline. Then we start with stage S_1 again. We call the proposed algorithm GREEDY-B in the following.

To prove the optimality of GREEDY-B, we introduce a few definitions: during the whole execution, for $1 \leq i \leq n$, $nbout$ is the total number of data sets that are output; $nbst_i$ is the number of setups performed on stage S_i ; $nbst = \sum_{i=1}^n nbst_i$ is the total number of setups; and $nbcomp_i$ is the average number of data sets processed between two setups on stage S_i . We have for $1 \leq i \leq n$:

$$nbcomp_i = \frac{nbout}{nbst_i}, \quad nbst_i = \frac{nbout}{nbcomp_i}, \quad \text{and} \quad nbst = \sum_{i=1}^n \frac{nbout}{nbcomp_i}.$$

Proposition 1. *For each stage S_i ($1 \leq i \leq n$), $nbcomp_i \leq b$.*

Proof. For each stage S_i , the number of data sets that can be processed after a setup is limited by its surrounding buffers. Once a setup is done to any stage S_i , it is not possible to perform more computations than there are data sets or than there is room for result sets. Since all buffers can contain exactly b data sets, we have $nbcomp_i \leq b$.

Proposition 2. *On a single processor with homogeneous buffers, the period can be expressed as $\mathcal{P} = \sum_{i=1}^n \frac{w_i}{v} + \sum_{i=1}^n \frac{st_i}{nbcomp_i}$.*

Proof. The period is the total execution time divided by the total number of processed data sets $nbout$. The execution time is the sum of the time spent computing, and the time to perform the setups. The computation time is the time to compute each stage once (w_i/v for stage S_i), multiplied by the number of data sets $nbout$. The reconfiguration time is the sum of the times required to perform each setup: $nbst_i \times st_i$. Therefore, the period can be expressed as $\mathcal{P} = \frac{1}{nbout} (\sum_{i=1}^n \frac{w_i}{v} \times nbout + \sum_{i=1}^n st_i \times nbst_i)$, and we conclude the proof by stating that $nbst_i = \frac{nbout}{nbcomp_i}$.

Lemma 1. *On a pipeline with homogeneous buffers, the lower bound of the period on a processor is $\mathcal{P}_{min} = \sum_{i=1}^n \frac{w_i}{v} + \sum_{i=1}^n \frac{st_i}{b}$.*

Proof. The result comes directly from Propositions 1 and 2:

$$\mathcal{P} = \sum_{i=1}^n \frac{w_i}{v} + \sum_{i=1}^n \frac{st_i}{nbcomp_i} \geq \sum_{i=1}^n \frac{w_i}{v} + \sum_{i=1}^n \frac{st_i}{b} = \mathcal{P}_{min}.$$

Theorem 1. *The scheduling problem on a single processor can be solved in polynomial time, using the GREEDY-B algorithm.*

Proof. It is easy to see that GREEDY-B is always performing b computations between two setups, and therefore $nbcomp_i = b$ for $1 \leq i \leq n$. Therefore, the period obtained with this algorithm is exactly \mathcal{P}_{min} , which is a lower bound on the period and hence it is optimal.

Multi Processor Scheduling ($b_i = b_u$). The interval mapping problem on fully homogeneous platforms without setup times can be solved in polynomial time using dynamic programming [13,14]. We propose the use of this dynamic programming algorithm for homogeneous platforms, taking into account the setup times in the calculation of a processor's period. To be precise, the calculation of the period is the one obtained by the GREEDY-B algorithm.

Let $c(j, k)$ be the optimal period achieved by any interval mapping that maps stages S_1 to S_j and that uses at most k processors. Let $per(i, j)$ be the average period of the processor on which stages S_i to S_j are mapped. Note that $per(i, j)$ takes the communication step into account. We have:

$$c(j, k) = \min_{1 \leq l \leq j-1} (\max(c(l, k-1), per(l+1, j))),$$

with the initial condition $c(j, k) = +\infty$ if $k > j$. Given the memory M , we can compute the corresponding buffer capacity $b(i, j) = \left\lfloor \frac{M}{\sum_{k=i}^{j+1} \delta_k} \right\rfloor = b_u$, since we assume identical buffer capacities. Therefore:

$$per(i, j) = \max \left(\frac{\delta_i}{\beta}, \sum_{k=i}^j \left(\frac{w_k}{v} + \frac{st_k}{b(i, j)} \right), \frac{\delta_{j+1}}{\beta} \right)$$

The main difference with the ordinary use of the dynamic programming algorithm is that P_u consumes b_u input data sets or outputs b_u data sets in waves because of GREEDY-B. So $c(n, p)$ returns the optimal period if and only if the period is actually dictated by the period of the slowest processor, i.e., the slowest processor cannot be in starvation or in saturation because of intermittent access to the input/output buffers. The following theorem ensures that this is true:

Theorem 2. *On a pipeline with inner-processor homogeneous buffer capacities b_u , the period \mathcal{P} is dictated by the period of the slowest processor.*

The proof can be found in the companion research report [3]. It is a proof by induction, and several cases need to be discussed considering a pipeline of processors: we prove that the slowest of the processors is never slowed down either by a lack of data inputs or by a saturation of its output buffer.

Single Processor Scheduling with Different Buffer Sizes. We complete the fixed buffer size study by considering buffers with different sizes. GREEDY-B chooses either a stage whose input buffer is full and we have enough space to fully empty it, or a stage whose output buffer is empty and we have enough data sets to compute in order to fully fill it. That way, we still maximize the amount of data sets processed after each setup: we are limited by the lowest capacity buffer, which is either a fully emptied input buffer, or a fully filled output buffer. It may not return an optimal schedule in the general case, but we can prove its optimality in the case of *multiple buffers*, i.e., each buffer capacity is a multiple of the capacities of both its predecessor and its successor: for $1 \leq i \leq n$, $\min(b_i, b_{i+1}) | \max(b_i, b_{i+1})$.

Theorem 3. *The scheduling problem with multiple buffers on a single processor can be solved in polynomial time, using the GREEDY-B algorithm.*

The proof of this theorem can be found in the companion research report [3]. Note that GREEDY-B is not optimal for multiple processor scheduling with multiple buffers.

4 Variable Buffer Sizes

In this section, we tackle the problem of allocating the buffers for all stages on a single processor P from an available memory M . We first focus on platforms with homogeneous data input sizes ($\delta_i = \delta$) and setup times ($st_i = st$).

Allocation Algorithm. If n stages are mapped on one processor then it needs $n + 1$ buffers. Given the memory M and the size of the data δ , if we want all buffers to contain the same number of data sets, then the maximum number of data sets that can fit in each buffer can be computed as $b = \left\lfloor \frac{M}{(n+1)\delta} \right\rfloor$.

The ALL-B algorithm allocates memory for each buffer according to this uniform distribution. The actual memory allocated for each buffer is $m_i = m = b\delta = \left\lfloor \frac{M}{n+1} \right\rfloor$. The memory used by this allocation is then $(n + 1)\delta \times b \leq M$, and we call $\mathcal{R} = M - (n + 1)\delta \times b$ the *remainder* of memory after the allocation, i.e., the unused part of the memory. We prove that this allocation algorithm is optimal if the remainder is lower than δ .

Theorem 4. *The algorithm ALL-B is optimal on a single processor (i.e., the period is minimized with this allocation) when $\mathcal{R} = M - (n + 1)\delta \times \left\lfloor \frac{M}{(n+1)\delta} \right\rfloor < \delta$.*

The proof can be found in the companion research report [3]. It is a proof by induction on n , and by expressing the general period (with any buffer sizes), we prove that the minimum is reached when all buffers are identical. The idea behind this proof is that, starting from a uniform allocation (same buffer sizes), raising the size of a buffer means reducing the size of another. The period is based on the amount of computations done before a setup (the $\frac{st}{\min(b_i, b_{i+1})}$ part of the period), and this value depends on the minimum of two consecutive buffers. Therefore we would need to raise more buffers than we lower to balance this value.

Memory Remainder. If there is a remainder in the memory after the allocation of buffers ALL-B, it is under certain conditions possible to use this remainder to increase the size of some buffers. It may also be possible to have another allocation, not based on ALL-B, that would make better or full use of the memory. In both cases, the period achieved by some scheduling algorithm may be lower than the one we have.

Proposition 3. *Given an application with homogeneous setup times st and input sizes δ , ALL-B may not give an optimal solution if $\mathcal{R} \geq \delta$.*

Proof. Let us consider a single processor, with a memory $M = 20$, and a speed $v = 1$. A total of $n = 6$ stages are mapped on this processor, and we have $\delta = w = st = 1$. There are seven buffers, and therefore ALL-B returns buffers of size $b = 2$, and the remainder is $\mathcal{R} = 20 - 2 \times 7 = 6$. The optimal period using this distribution is obtained by scheduling the stages with the GREEDY-B algorithm (see Theorem 1), and therefore:

$$\mathcal{P} = \sum_{i=1}^6 \frac{w_i}{v} + \sum_{i=1}^6 \frac{st}{b} = 6 + \left(\frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} \right) = 9.$$

However, let us consider the following allocation: $b_1 = b_2 = b_3 = b_4 = 2$ and $b_5 = b_6 = b_7 = 4$. This allocation uses all the memory, and it corresponds to the definition of *multiple buffers*. Therefore, the optimal period is obtained by scheduling the stages with the GREEDY-B algorithm, and:

$$\mathcal{P} = \sum_{i=1}^6 \frac{w_i}{v} + \sum_{i=1}^6 \frac{st}{\min(b_i, b_{i+1})} = 6 + \left(\frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} \right) = 8.5.$$

This allocation leads to a smaller period than ALL-B, which concludes the proof.

We propose an heuristic to deal with the memory remainder created by ALL-B ($\forall 1 \leq i \leq n + 1, b_i = b$). In some cases, it is possible to use \mathcal{R} to increase the size of several (but not all) buffers. According to Proposition 3, the use of this remainder may lead to a decrease of the period. We restrict to the construction of *multiple buffers* as defined above, so that we are able to find optimally the period thanks to the GREEDY-B algorithm. Hence, if there is enough memory to increase the size of buffers by steps of b , and if there is at least $2b\delta$ memory left, then the size of two consecutive buffers can be doubled, resulting in halving the number of setups for the corresponding stage.

The heuristic, that we call H-REMAIN, starts off by doubling the size of the two last buffers if there are $2b\delta$ memory units left, then will continue to increase the capacity of the adjacent buffers by b as long as $b\delta$ memory units are still available. Note that since $\mathcal{R} < (n + 1)\delta$, the algorithm is guaranteed to end before having doubled the size of all buffers.

Given the available memory M , $\mathcal{P}_b(M)$ is the period obtained if $\forall i \in [1, n + 1], b_i = b$; $\mathcal{P}_{algo}(M)$ is the period obtained by our heuristic; and $\mathcal{P}_{opt}(M)$ is the optimal (minimal) period that can be achieved with memory M .

We compute the value of b obtained by the ALL-B algorithm, and therefore $M = b(n + 1)\delta + \mathcal{R}$, with $\mathcal{R} < (n + 1)\delta$. It has already been proved (see Theorem 4) that if there is no remainder after ALL-B, $\mathcal{P}_b(M)$ is optimal. More formally, $M = b(n + 1)\delta \iff \mathcal{P}_b(M) = \mathcal{P}_{opt}(M)$. We define $M^* = (b + 1)(n + 1)\delta = M + (n + 1)\delta - \mathcal{R}$. With a memory M^* , there is also no remainder and $\mathcal{P}_{b+1}(M^*) = \mathcal{P}_{opt}(M^*)$. We first prove that both $\mathcal{P}_{algo}(M)$ and $\mathcal{P}_{opt}(M)$ can be bounded by $\mathcal{P}_b(M)$ and $\mathcal{P}_{b+1}(M^*)$ respectively:

Lemma 2. *We have $\mathcal{P}_b(M) \geq \mathcal{P}_{algo}(M) \geq \mathcal{P}_{opt}(M) \geq \mathcal{P}_{b+1}(M^*)$.*

Proof. By definition, we have $\mathcal{P}_{algo}(M) \geq \mathcal{P}_{opt}(M)$. For the upper bound, H-REMAIN is potentially improving $\mathcal{P}_b(M)$ by exploiting the remainder, and the period cannot be increased by the allocation of the remainder of the memory.

For the lower bound, note that $\mathcal{P}_{b+1}(M^*)$ is the optimal period with memory $M^* > M$, and therefore $\mathcal{P}_{opt}(M)$ cannot be better, otherwise we would have a better solution with M^* that would not use all memory.

Theorem 5. *The two algorithms ALL-B and H-REMAIN are $\frac{b+1}{b}$ -approximation algorithms.*

Proof. Let $W = \sum_{i=1}^{n+1} \left(\frac{w_i}{v}\right)$. We have $\mathcal{P}_b(M) = W + \frac{(n+1)st}{b}$, and $\mathcal{P}_{b+1}(M^*) = W + \frac{(n+1)st}{b+1}$. Therefore,

$$\frac{\mathcal{P}_b(M)}{\mathcal{P}_{b+1}(M^*)} = \frac{W + \frac{(n+1)st}{b}}{W + \frac{(n+1)st}{b+1}} \leq \frac{\frac{(n+1)st}{b}}{\frac{(n+1)st}{b+1}} = \frac{b+1}{b},$$

since $W > 0$ and $\frac{(n+1)st}{b+1} \leq \frac{(n+1)st}{b}$. Finally, thanks to Lemma 2, we have:

$$\mathcal{P}_{algo}(M) \leq \mathcal{P}_b(M) \leq \frac{b+1}{b} \mathcal{P}_{b+1}(M^*) \leq \frac{b+1}{b} \mathcal{P}_{opt}(M),$$

which concludes the proof (recall that $\mathcal{P}_b(M)$ is the period obtained by algorithm ALL-B). Note that the worst approximation ratio is achieved for $b = 1$, and then we have 2-approximation algorithms. However, when b increases, the period achieved by the algorithms tend to the optimal solution.

With Heterogeneous Setup Times or Data Input Sizes (st_i, δ_i). The case of heterogeneous setup times (st_i) is kept for future work, since it turns out to be much more complex. Indeed, allocating buffers while taking setup times into account requires to prioritize higher setup times by allocating larger buffer capacities. However, this requires both the input and output buffers of the corresponding stage to be larger, and it will inevitably lead to side effects on surrounding stages.

For heterogeneous data input sizes (δ_i), we can use a variant of the ALL-B algorithm to allocate buffers of identical capacities, in terms of data sets: $b_i = \left\lfloor \frac{M}{\sum_{k=1}^{n+1} \delta_k} \right\rfloor = b$. In this case, the memory used is $\sum_{i=1}^{n+1} b \times \delta_i \leq M$, and the remainder is $\mathcal{R} = M - \sum_{i=1}^{n+1} b \times \delta_i$. However, even if there is no remainder, the allocation may not be optimal:

Let us consider a single processor, with a memory $M = 301$, speed $v = 1$. There are $n = 4$ stages with $w = st = 1$. The different input sizes are: $\delta_1 = 20, \delta_2 = 20, \delta_3 = 1, \delta_4 = 1, \delta_5 = 1$. ALL-B returns buffers of size $b = 7$, and the remainder is $\mathcal{R} = 301 - (20 \times 7 + 20 \times 7 + 1 \times 7 + 1 \times 7) = 0$. The optimal period using this distribution is obtained by scheduling the stages with the GREEDY-B algorithm (see Theorem 1), and therefore:

$$\mathcal{P} = \sum_{i=1}^4 \frac{w_i}{v} + \sum_{i=1}^4 \frac{st}{b} = 4 + \left(\frac{1}{7} + \frac{1}{7} + \frac{1}{7} + \frac{1}{7} \right) = 4.571.$$

However, let us consider the following allocation: $b_1 = b_2 = 6$ and $b_3 = b_4 = b_5 = 18$. This allocation uses less memory, yet has way higher capacity buffers for b_3 to b_5 , with the only trade-off being the reduction of the capacity of b_1 and b_2 by one. This allocation corresponds to the definition of multiple buffers. Therefore, the optimal period is obtained by scheduling the stages with GREEDY-B, and $\mathcal{P} = \sum_{i=1}^4 \frac{w_i}{v} + \sum_{i=1}^4 \frac{st}{\min(b_i, b_{i+1})} = 4 + \left(\frac{1}{6} + \frac{1}{6} + \frac{1}{18} + \frac{1}{18} \right) = 4.444$.

This allocation leads to a smaller period than ALL-B.

5 Conclusion

In this paper, we present solutions to the problem of optimizing setup times and buffer use for pipeline workflow applications. For the problem of fixed buffer sizes of identical size within a same processor, we provide an optimal greedy algorithm for a single processor, and a dynamic programming algorithm for multiple processors. In the latter case, the application period is equal to the period of the slowest processor. In the case of variable buffer sizes, we tackle the problem of distributing the available processor memory into buffers such that the period is minimized. When the memory allocation results in no remainder (the whole memory is used), the algorithm turns out to be optimal, and we propose some approximation algorithms for the other cases.

In future work, we plan to consider sequence-dependent setup times ($st_{i,j}$), a problem that is already known to be NP-complete. We envisage the design of competitive heuristics, whose performance will be assessed through simulation. Furthermore, for the st_i case, we plan to investigate the memory allocation problem on a single processor. On the long term, we will consider the case of heterogeneous buffer capacities b_i . This case is particularly interesting, as the buffer allocation heuristics lead to heterogeneous buffer sizes.

References

- Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.: A survey of scheduling problems with setup times or costs. *European J. of Op. Research* 187(3), 985–1032 (2008)
- Allahverdi, A., Soroush, H.: The significance of reducing setup times/setup costs. *European Journal of Operational Research* 187(3), 978–984 (2008)
- Benoit, A., Coqblin, M., Nicod, J.M., Philippe, L., Rehn-Sonigo, V.: Throughput optimization for pipeline workflow scheduling with setup times. *Research Report 7886, INRIA* (2012), <http://graal.ens-lyon.fr/~abenoit/papers/RR-7886.pdf>
- Benoit, A., Robert, Y.: Mapping pipeline skeletons onto heterogeneous platforms. *J. Parallel and Distributed Computing* 68(6), 790–808 (2008)
- Bhat, P., Raghavendra, C., Prasanna, V.: Efficient collective communication in distributed heterogeneous systems. In: 19th ICDCS 1999, pp. 15–24 (1999)
- Bhat, P., Raghavendra, C., Prasanna, V.: Efficient collective communication in distributed heterogeneous systems. *JPDC* 63, 251–263 (2003)
- Norman, B.A.: Norman: Scheduling flowshops with finite buffers and sequence-dependent setup times. *Comp. & Indus. Engineering* 36(1), 163–177 (1999)
- Gendreau, D., Gauthier, M., Hériban, D., Lutz, P.: Modular architecture of the microfactories for automatic micro-assembly. *Journal of Robotics and Computer Integrated Manufacturing* 26(4), 354–360 (2010)
- Li, L., Qiao, F.: Aco-based scheduling for a single batch processing machine in semiconductor manufacturing. In: IEEE Int. CASE 2008, pp. 85–90 (2008)
- Li, L., Qiao, F., Wu, Q.: Aco-based scheduling of parallel batch processing machines to minimize the total weighted tardiness. In: Int. CASE 2009, pp. 280–285 (2009)
- Luh, P.B., Gou, L., Zhang, Y., Nagahora, T., Tsuji, M., Yoneda, K., Hasegawa, T., Kyoya, Y., Kano, T.: Job shop scheduling with group-dependent setups, finite buffers, and long time horizon. *Annals of Operations Research* 76, 233–259 (1998)

12. Srikar, B., Ghosh, S.: A milp model for the n-job, m-stage flowshop with sequence dependent set-up times. *Int. J. of Production Research* 24(6), 1459–1474 (1986)
13. Subhlok, J., Vondran, G.: Optimal mapping of sequences of data parallel tasks. *ACM SIGPLAN Notices* 30(8), 134–143 (1995)
14. Subhlok, J., Vondran, G.: Optimal latency-throughput tradeoffs for data parallel pipelines. In: *Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, p. 71. ACM (1996)
15. Zhang, M., Goldberg, K.: Calibration of wafer handling robots: A fixturing approach. In: *IEEE Int. CASE 2007*, pp. 255–260 (2007)

Meteorological Simulations in the Cloud with the ASKALON Environment

Gabriela Andreea Morar¹, Felix Schüller², Simon Ostermann³,
Radu Prodan³, and Georg Mayr²

¹ Babes-Bolyai University, Cluj-Napoca, Romania
`gabriela.morar@econ.ubbcluj.ro`

² Institute for Meteorology and Geophysics, University of Innsbruck
`{felix.schueller,georg.mayr}@uibk.ac.at`

³ Institute of Computer Science, University of Innsbruck
`{simon,radu}@dps.uibk.ac.at`

Abstract. Precipitation in mountainous regions is an essential process in meteorological research for its strong impact on the hydrological cycle. To support scientists, we present the design of a meteorological application using the ASKALON environment comprising graphical workflow modeling and execution in a Cloud computing environment. We illustrate performance results that demonstrate that, although limited by Amdahl’s law, our workflow can gain important speedup when executed in a virtualized Cloud environment with important operational cost reductions. Results from the meteorological research show the usefulness of our model for determining precipitation distribution in the case of two field campaigns over Norway.

1 Introduction

Scientific computing requires an ever-increasing number of resources to deliver results for growing problem sizes in a reasonable time frame. Today, Cloud computing provides an alternative by which parallel resources are no longer hosted by the researcher’s computational facilities or shared as in computational Grids, but leased from large specialized data centers only when needed. To account for the heterogeneity and loosely coupled nature of resources, the scientific community adopted the workflow paradigm based on loosely coupled coordination of atomic activities as one of the most successful programming paradigms. As a consequence, numerous efforts among which the ASKALON environment [1] developed integrated environments to support the development and execution cycle of scientific workflows on dynamic Grid and Cloud environments. In this paper, we illustrate a case study of using ASKALON for porting and executing a meteorological application in a real Cloud environment. First of all, the application is specified by the user at a high-level of abstraction using a graphical UML modeling tool or an XML-based specification language. A set of advanced middleware services consisting of resource management, scheduling, and enactment

support the transparent execution of the application on the underlying Cloud resources.

The paper is organized as follows. Section 2 gives a small introduction to the meteorological goals of our interdisciplinary research. Section 3 gives an overview of the ASKALON environment used to program and execute the meteorological application on Cloud computing resources with improved performance. Section 4 presents the workflow engineering details using the ASKALON graphical user interface. Section 5 presents performance and output research results from running our application on an academic private Cloud infrastructure. Section 6 summarizes the related work and Section 7 concludes the paper.

2 Meteorological Research

The aim of our meteorological research is to investigate and simulate precipitation in mountainous regions with a simple meteorological numerical model called linear model of orographic precipitation (LM) [2]. Applications of this model range from climatological studies to hydrological aspects. As LM is a very simple and basic model, it can be easily run in a large number of parameter studies. The model works on gridded topography and gives an analysis of precipitation resulting from a given meteorological setup. The topography can be realistic like the Alps or idealized shapes for research purposes. In the LM, a moist air mass is forced over a given topography. Lifting, which is described by simple nonhydrostatic airflow dynamics, produces cloud water. It is converted to hydrometeors and drifted with adjustable timescales.

Compared to more complex models, one big advantage of LM is the ease with which a higher horizontal resolution of the results can be achieved. Furthermore, as it is simple, a single run executes very fast compared to more complex models (i.e. in the order of minutes). Due to the fast execution time, LM allows for 1000+ instances in a short amount of time, allowing us to quickly check model sensitivity through a range of parameters or to make probabilistic forecasts by having a statistically significant number of experiments. Furthermore, no communication between the parallel instances is needed, making it an ideal application for the high latency setup of a Cloud environment, opposed to a full fledged meteorological model where a more tightly coupled parallelization (e.g. MPI) is needed. There are relatively few requirements for the parallel instances, since they only need to read small input files (a few megabytes). However the workflow activities that are collecting and extracting the results need considerably more storage I/O. The amount of data from the parallel instances needs to be condensed to be of use for a meteorological end user.

Our intent is to use the LM applications in two main areas. (1) *Research area* concentrates on investigation of the model behavior by extending the LM theory. The experiments in this area need to be very flexible, as for example not all parts of the workflow are necessarily run at each invocation. One main use is to investigate the model behavior after new theories are implemented into the model. Another option is to use it as a tool for analysis of certain meteorological phenomena or meteorological measurements. These experiments vary

in the requirements to the computational infrastructure, as they vary in size, data-wise and computational-wise. The experiments of the research type are run on-demand. (2) *The operational area* aims at providing the Tyrolean avalanche service (“Tiroler Lawinenwarndienst” (LWD)) with a downscaled and probabilistic precipitation forecast, helping them with their daily issued avalanche bulletin. Taking input from a global numerical forecast model, we downscale these results to a higher horizontal resolution and test the sensitivity of the results to changes in the input parameters. The results are visualized as probability maps for the area of Tyrol. Requirements for this aspect are the availability and the reliability of the compute infrastructure. Experiments of the operational type are only run once every day for a short time, making the Cloud a more economical way to run the computations as compared to the costs of a dedicated system.

3 ASKALON

ASKALON [1] is an application development and computing environment developed at the University of Innsbruck with the goal of simplifying the development and optimization of applications that can harness the power of Grid and Cloud computing infrastructures (see Figure 1). In ASKALON, the user composes workflow applications at a high level of abstraction using a UML graphical modeling tool. Workflows are specified as a directed graph of *activity types* representing an abstract semantic description of the computation such as a Gaussian elimination algorithm, a fast Fourier transform, or an N-body simulation. The activity types are interconnected in a workflow through control flow and data flow dependencies. The abstract workflow representation is given in an XML form to the ASKALON middleware services for transparent execution onto the Grid/Cloud.

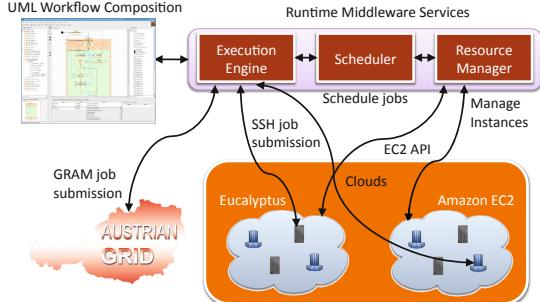


Fig. 1. Simplified ASKALON architecture extended for computational Clouds

4 RainCloud Workflow

To be able to run the original meteorological application on a distributed Grid/Cloud infrastructure, we split the monolithic simulation code in a workflow called RainCloud consisting of a set of activities described next.

4.1 Graphical Modeling

The workflow illustrated in Figure 2 receives two files as input data to be processed: `Topography.tar.gz` and `DataIN.tar.gz`. The `PrepareLM` activity has to be run by each workflow invocation and does all preprocessing of the input data. The exact nature of this depends on the flavor of the workflow invocation (see Section 4.2). Often this activity loops through ranges of model parameters and/or processes, given a topography and meteorological input data. The results of this activity are the number of `LinearModel` instances (saved in `Template_iterations.txt`) needed for future processing and the base files used for the linear model (contained in `PLM_g_out.tar.gz`). `LinearModel` is the main activity of the workflow where the meteorological model is being run as a parallel loop. `LinearModel` takes input produced by `PrepareLM` consisting of topography and meteorological control parameters and outputs a precipitation analysis. Based on the generated number of `LinearModel` instances, the workflow creates the corresponding number of parallel activities (around 1000 for a realistic simulation). To decrease the overhead for creating a large number of parallel activities and for transferring the corresponding files, we created an input parameter called `NGroup` that defines the number of linear model processes to be grouped into one `LinearModel` activity. The output of the `LinearModel` activities are `LM_g_out.tar.gz` files that can be further processed by other activities, such as `PostprocessSingle` (an optional part of the workflow) which extracts certain points from the result, condenses them to sums/means, or performs postprocessing of the results from the linear model where no comparison/combination with other model runs is needed. This activity is also used for parts of the visualization. The `LM_g_out.tar.gz` files can be further postprocessed by the `PostprocessFinalLM` activity that combines the output from different linear model runs or generates a visualization of the raw model output. The `PostprocessSingle` activity outputs `PPS_g_out.tar.gz` files that are further used as input for the `PostprocessFinalPPS` activity, which combines

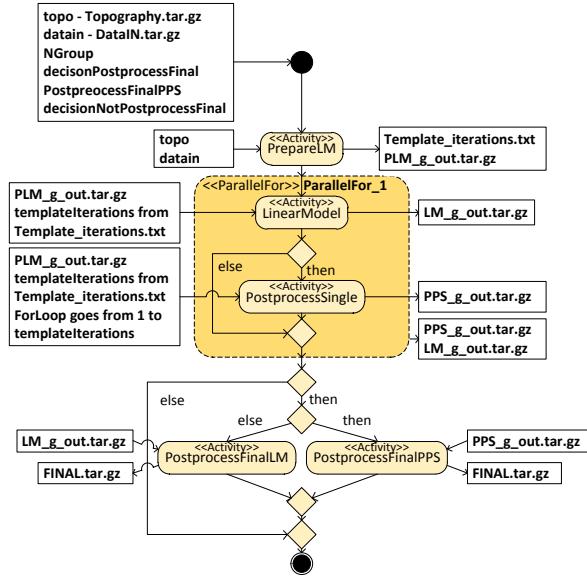


Fig. 2. Graphical representation of the workflow in ASKALON

the results from `PostprocessSingle`, performs result correction/normalization, and creates a visualization of the already postprocessed data. The output of `PostprocessFinalLM` and `PostprocessFinalPPs` is the `FINAL.tar.gz` file that contains the final results.

4.2 Workflow Flavors

We designed the workflow to be run in three “flavors”. *Ideal* belongs to the operational area in which the whole setup uses idealized topography and atmospheric conditions. This workflow is mostly used for model testing or investigation of meteorological phenomena; *Semi-ideal* belongs to the research area and is designed such that either the topography is idealized or the atmospheric input is simplified, however, at least one part has real world application. For example, the topography is based on real topography data, but the atmospheric conditions are “manually” given. This flavor is used for e.g. interpretation of meteorological measurements; *Real* belongs to the research area such that both topography and atmospheric conditions are given by real-world observations or full numerical models. This flavor is used for forecasting/downscaling precipitation.

We designed the workflow such that it satisfies the requirements of the three flavors through the use of three decision nodes based of three input parameters: `decisionPostprocessSingle` – if true then the `PostprocessSingle` activity gets executed; `decisonPostprocessFinal` – if true then the `PostpreocessFinalPPs` activity gets executed, otherwise `PostprocessFinalLM`; `decisionNotPostprocessFinal` – if true then one of the `postprocessfinal` activities will get executed based on the evaluation of `decisonPostprocessFinal`.

5 Experiments

The goal of our experiments is twofold: (1) to investigate whether a parallel Cloud computing infrastructure is beneficial to the RainCloud workflow execution in terms of performance and costs; (2) to apply the RainCloud workflow to achieve our meteorological goals in research and operational areas.

5.1 Performance Results

The purpose of the performance experiments was to study the performance we can obtain by running the RainCloud workflow in various virtual machine configurations and on a public Cloud. We use a private Cloud that runs Eucalyptus and is based on four machines, each equipped with two Intel Xeon X5570 quad-core processors and 32 GB of main memory. Therefore our entire infrastructure provides 32 cores and 128 GB of main memory of which 112 GB are available for the Cloud. We run the workflow using different virtual machine instance types characterized by different numbers of cores (1, 2, 4, 8) and amount of memory (2, 4, 8, and 16 GB). The instance types are named from A–D with increasing numbers of cores and memory size. We continue our experiments with running the workflow on Amazon EC2 using 1 to 4 instances of the type c1.xlarge.

Throughout the rest of the paper PS1 and PS2 denote the two different problem sizes used in our experiments, with PS2 holding approximately double the computational work of PS1. In meteorological terms the different problem sizes investigate different ranges and resolutions of the targeted parameter space, in this case temperature at the ground and layer interface height of atmospheric levels.

Before performing the speedup analysis, we must emphasize that the workflow has a sequential part that limits the maximum speedup according to the Amdahl's law. Figure 3 presents a sample workflow trace snapshot obtained from the ASKALON performance analysis tool which shows that the first sequential activity takes 70 seconds and the last sequential activity 50 seconds from a total of 313 seconds of run time. This leads to the observation that for this execution, the sequential part accounts for over a third of the overall run time using 24 cores distributed to three instances.

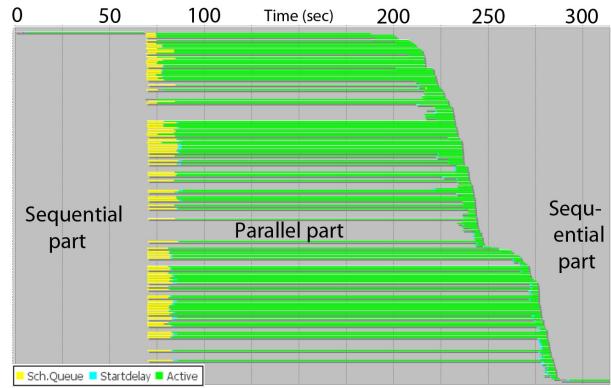
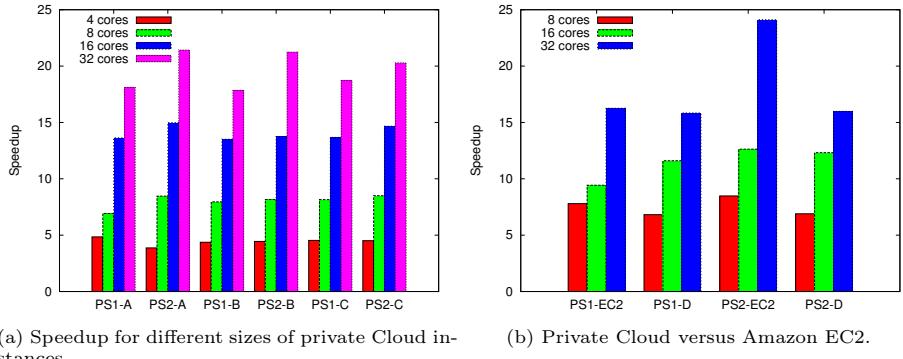


Fig. 3. Workflow execution trace snapshot using the ASKALON performance tool

Figure 4b shows the speedup of the workflow when using 1, 2 or 4 Cloud instances. We compare the performance of our private Cloud installation with configuration D (8 cores and 16 GB of memory per instance) with the results when using the Amazon EC2 c1.xlarge instances (8 cores and 7 GB of memory). The Speedup was computed using the serial execution time obtained by running the workflow on 1 core instances for the private and the public Cloud. The speedup compared to a sequential execution reaches a factor of 23 for PS2 and 32 cores of the Amazon Cloud. Our evaluation shows that the performance of the private Cloud installation is comparable with that of the public environment. This allows us to use the private Cloud for development and we can then easily switch to the public Cloud once the production runs begin.

Figure 4a depicts the speedup gained by using different Cloud instance types with a fixed ratio of 2GB memory per core, where the sequential time was computing by running the workflow on a one-core instance using the corresponding amount of memory. The experiments were run for both problem sizes.

As illustrated, the results show that the application is scalable, reaching a maximum speedup of 21.4 for PS2 with instance type A and 32 cores. In addition the results indicate that the application scales better for higher problem sizes, which was the case for almost all our experiment runs. The maximum speedup



(a) Speedup for different sizes of private Cloud instances.
(b) Private Cloud versus Amazon EC2.

Fig. 4. Speedup analysis. PS# denotes the problem size and A–D and EC2 the instance type.

increase of PS2 over PS1 is 22%; the average increase is 4.3%. As Figure 4a illustrates, varying the instance types only has a small effect on the speedup, with a maximum speedup increase of 20% for instance type C over D for PS1 using 8 cores, and 9% for instance type A over B for PS2 using 16 cores.

In general, the execution of one workflow with the experimental input data would cost approximately \$2.72 if executed on Amazon EC2 using 16 c1.medium instances (\$0.17/hour). This result applies to all presented workflow instances, as their execution time is lower than the one-hour payment granularity of EC2. For a yearly cost of \$992.8 this workflow can be run once every day, which is only a fraction of the amount the purchase of a comparable, dedicated system would cost.

5.2 Meteorological Results

In this section we show the meteorological results obtained by conducting two experimental runs using the RainCloud workflow: (1) an explorative run for optimizing a future experiment setup, and (2) a simulation of precipitation in the region of the Kongsvegen glacier, Svalbard, Norway.

Explorative Study. Extensive precipitation measurements taken during the 2006 STOPEX2 field campaign over Stord on the west coast of Norway provide ground truth for numerical simulation studies. In order to determine how large the spatial domain of these simulations has to minimally be in order to successfully simulate the flow and precipitation amounts, an explorative study investigates how strongly the precipitation of Stord is influenced by the downstream main “ridge” of Norway. For this experiment, we used a pseudo-three-dimensional cross section through the center of Stord using the ideal RainCloud workflow flavor (see Figure 5(a)). We run the model with two atmospheric layers where winds differ. We changed the length of the topography within each of these experiments

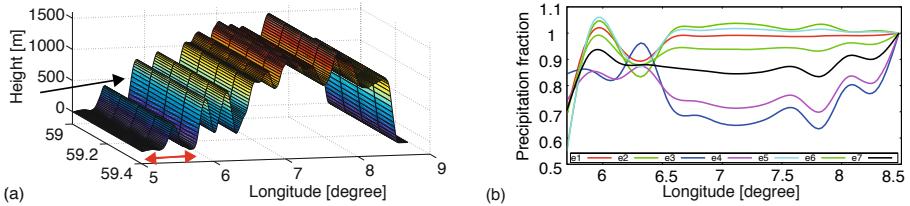


Fig. 5. (a) Topography [meters] used for the explorative study. The black arrow indicates wind direction, the red arrow the target area of Stord. Latitude is degrees north, longitude degrees east. (b) Precipitation sum over Stord (5.5–5.7 degrees) from varying topography lengths as fraction of the reference precipitation. The experiments are (upper level wind speed/lower level wind speed in m/s): e1 (15/15), e2 (5/15), e3 (45/15), e4 (45/45), e5 (5/5), e6 (25/25), e7 (35/35).

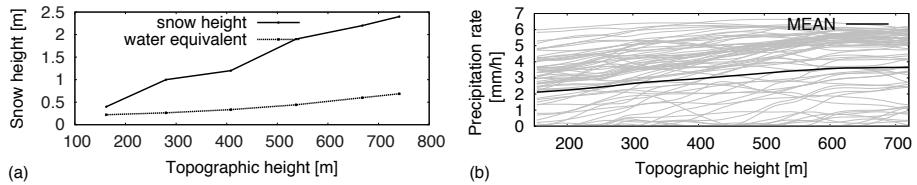


Fig. 6. (a) Measured snow height and water equivalent along topographic height of the Kongsvegen glacier during winter 2010/2011. (b) Precipitation rates from LM simulations with varying wind directions and speeds in respect to topographic height.

from only having Stord (5.5–5.7 deg E) up to the full reference length (5.5–8 deg E). Both variations lead to 690 instances of the activity `LinearModel` with a small input size per instance. Activity `PostProcessSingle` extracts a precipitation sum over the area of Stord between 5.5 and 5.7 deg E for each model run, which is then compared with the precipitation amount from the reference run in `PostprocessFinalPPS`. A subset of simulations is shown in Figure 5(b). Higher wind speeds lead to an increasing importance of the downstream mountains. For small wind velocities, we could cut the topography at around 6.8 deg longitude, accepting a small error of about 2 percent. When higher wind speeds occur, the complete Norwegian ridge needs to be included in activity `LinearModel`.

Snow Accumulation over a Norwegian Glacier. The research workflow setup was further used for an investigation of precipitation/snow accumulation on the Kongsvegen glacier on Svalbard, Norway (SvalClac project). Snow heights and water equivalents were derived from snow density measurements within snow pits on the central flow line of the Kongsvegen glacier. These measurements show a nearly linear gradient with topographic height. The question to be answered with our workflow simulations was whether this gradient is explainable by orographic precipitation effects. A topography with 150m resolution and meteorological values from a typical precipitation event (5th Nov. 2010) are used. In a

first set of experiments we varied wind speed from 0m/s to 40m/s and direction from 0 and 360 degrees, leading to 900 instances of `LinearModel`. One instance has a relatively big input size of 20Mb due to the high-resolution topography. Figure 6 (b) shows the simulated precipitation along the glacier extracted by `PostProcessSingle` (no `PostprocessFinal` is run). The snow measurements are an “integral” over the whole winter including various wind directions and speed, therefore we apply a mean over all model results including those without precipitation (black line). It shows that the model exhibits a similar behavior as the measurements, indicating that the gradient is explainable by orographic precipitation effects. To eliminate other possible causes further investigation is necessary.

6 Related Work

The scientific community shows growing interest in Cloud computing. The FutureGrid [3] project provides a scientific Cloud infrastructure on demand for scientists to experiment with this environment without the cost that commercial Cloud providers charge for their usage. The infrastructure is only available for research purposes and production runs are strictly prohibited. The work in [4] shows that Cloud computing can be used for scientific workflows, with the Montage workflow. This proof of concept motivated the community to start exploring this resource type as a Grid alternative. Other approaches [5] use Cloud resources in a different way, e.g. by extending Clusters with additional resources from a Cloud during peak usage, increasing the throughput of the system. A similar extension of the Torque job manager to add Cloud resources to clusters is presented in [6]. Our approach does not rely on any existing clusters and tries to optimize the workflow execution on a Cloud-only environment. The research presented in [7] shows a workflow engine developed for Cloud computing. We have the advantage of a mature workflow engine, which evolved in the Grid area and was extended for a hybrid usage of both technologies. In this paper we use it with Clouds only to show that it is well optimized for this resource class as well. The Megha workflow management system [8] is designed to execute scientific applications, designed in the form of workflows described in xWFL, on Cloud and Grid resources. It is created in the format of a portal. The authors prove that by tuning the applications to run on Amazon EC2 cloud resources time consumption can be significantly reduced. Aneka [9] is a platform and a framework for developing distributed applications on the Cloud. It harnesses the spare CPU cycles of a heterogeneous network of desktop PCs and servers or data centers on demand. A parallel among the performance of execution of scientific workflows on commercial cloud resources (Amazon EC2) and HPC systems (NCSA’s cluster) is presented in [10]. Three different workflows are tested: Montage, Broadban and Epigenom. The performance is similar, although a bit lower for the Cloud, due to less powerful EC2 resources.

7 Conclusions

We illustrated a case study of using ASKALON for porting and executing a real-world meteorological application called RainCloud in an academic private Cloud environment. The application designed as a workflow implements a highly simplified linear model of orographic precipitation to answer meteorological research questions in connection with measurements from two field campaigns in mountainous areas. We presented the application design performed at a high-level of abstraction using a graphical modeling tool. A set of advanced middleware services comprising resource management, scheduling, and enactment support the transparent execution of the application on the underlying Cloud resources. We present performance results that demonstrate that, although limited by Amdahl's law, our workflow application can gain important speedup when executed in a virtualized Cloud environment with significant cost reductions if operated in a production environment. Performance results achieved by using a private Cloud and those attained with Amazon EC2 instances show that we can use the private Cloud for development purposes and then are able to switch to a public Cloud when production runs will begin. Results from the meteorological research show that our simplified model is a useful tool for determining possible causes for precipitation distribution in the case of two field campaigns over Norway.

Acknowledgment. The research was funded by the Standortagentur Tirol: RainCloud and data from the European Centre for Medium-Range Weather Forecasts was used for the meteorological application. G.A. Morar thanks the: Investing in people! Ph.D. scholarship, contract nr. POSDRU/88/1.5/S/60185.

References

1. Fahringer, T., Prodan, R., et al.: Askalon: A development and grid computing environment for scientific workflows. In: Scientific Workflows for Grids. Workflows for e-Science. Springer (2007)
2. Barstad, I., Schüller, F.: An Extension of Smith's Linear Theory of Orographic Precipitation: Introduction of Vertical Layers. *Journal of the Atmospheric Sciences* 68(11), 2695–2709 (2011)
3. Riteau, P., Tsugawa, M., Matsunaga, A., Fortes, J., Keahey, K.: Large-scale cloud computing research: Sky computing on futuregrid and grid'5000. *ERCIM News* (2010)
4. Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, G.B., Good, J.: On the use of cloud computing for scientific workflows. In: eScience, pp. 640–645. IEEE Computer Society (2008)
5. Assuncao, M., Costanzo, A., Buyya, R.: Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In: Kranzlmüller, D., Bode, A., Hegering, H.G., Casanova, H., Gerndt, M. (eds.) 11th IEEE International Conference on High Performance Computing and Communications, HPCC 2009. ACM (2009)
6. Marshall, P., Keahey, K., Freeman, T.: Elastic site: Using clouds to elastically extend site resources. In: CCGRID, pp. 43–52. IEEE (2010)

7. Franz, D., Tao, J., Marten, H., Streit, A.: A workflow engine for computing clouds. In: The Second International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2011, Rome, Italy, p. 6 (2011)
8. Pandey, S., Karunamoorthy, D., Gupta, K.K., Buyya, R.: Megha Workflow Management System for Application Workflows. IEEE Science & Engineering Graduate Research Expo (2009)
9. Vecchiola, C., Chu, X., Buyya, R.: Aneka: A Software Platform for .NET-based Cloud Computing. IOS Press (2010)
10. Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B.P., Maechling, P.: Scientific workflow applications on amazon ec2 (arXiv:1005.2718) (May 2010)

A Science-Gateway Workload Archive to Study Pilot Jobs, User Activity, Bag of Tasks, Task Sub-steps, and Workflow Executions

Rafael Ferreira da Silva and Tristan Glatard

University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France
`{rafael.silva,glatard}@creatis.insa-lyon.fr`

Abstract. Archives of distributed workloads acquired at the infrastructure level reputably lack information about users and application-level middleware. Science gateways provide consistent access points to the infrastructure, and therefore are an interesting information source to cope with this issue. In this paper, we describe a workload archive acquired at the science-gateway level, and we show its added value on several case studies related to user accounting, pilot jobs, fine-grained task analysis, bag of tasks, and workflows. Results show that science-gateway workload archives can detect workload wrapped in pilot jobs, improve user identification, give information on distributions of data transfer times, make bag-of-task detection accurate, and retrieve characteristics of workflow executions. Some limits are also identified.

1 Introduction

Grid workload archives [1–5] are widely used for research on distributed systems, to validate assumptions, to model computational activity [6, 7], and to evaluate methods in simulation or in experimental conditions. Available workload archives are acquired at the infrastructure level, by computing sites or by central monitoring and bookkeeping services. However, user communities often access the infrastructure through stacks of application-level middleware such as workflow engines, application wrappers, pilot-job systems, and portals. As a result, workload archives lack critical information about dependencies among tasks, about task sub-steps, about artifacts introduced by application-level scheduling, and about users. Methods have been proposed to recover this information. For instance, [8] detects bags of tasks as tasks submitted by a single user in a given time interval. In other cases, information can hardly be recovered: [2] reports that there is currently no study of a pilot-job workload, and workflow studies such as [5] are mostly limited to test runs conducted by developers.

Meanwhile, science gateways are emerging as user-level platforms to access distributed infrastructures. They combine a set of authentication, data transfer, and workload management tools to deliver computing power as transparently as possible. They are used by groups of users over time, and therefore gather rich information about workload patterns. The Virtual Imaging Platform (VIP) [9],

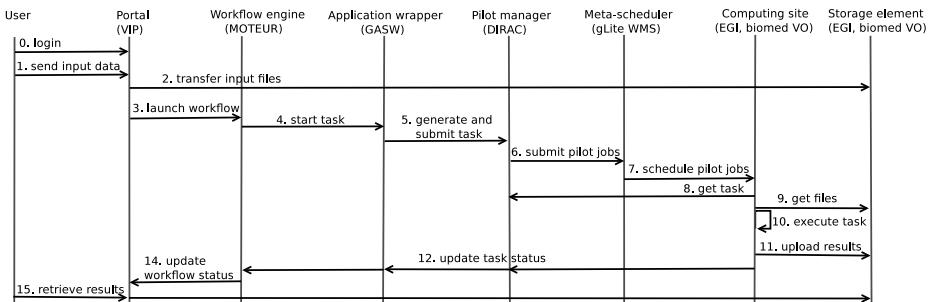


Fig. 1. Considered science-gateway architecture. Tools in brackets are used here.

for instance, is an open web platform for medical simulation. Other examples include e-bioinfra [10], the P-Grade portal [11], the Science-Gateway framework in [12], MediGRID [13], and CBRAIN¹.

This paper describes a science-gateway workload archive, and illustrates its added value to archives acquired at the infrastructure level. The model is presented in Section 2 and used in 5 case studies in Section 3: Section 3.1 studies pilot jobs, Section 3.2 compares user accounting to data acquired by the infrastructure, Section 3.3 performs fine-grained task analysis, Section 3.4 evaluates the accuracy of bag of task detection from infrastructure-level traces, and Section 3.5 analyzes workflows in production. Section 4 concludes the paper.

2 A Science-Gateway Workload Archive

Science gateways usually involve a subset or all the entities shown on Fig. 1, which describes the VIP architecture used here. Users authenticate to a web portal with login and password, and they are then mapped to X.509 robot credentials. From the portal, users mainly transfer data and launch workflows executed by an engine. The engine has an application wrapper which generates tasks from application descriptions and submits them to a pilot-job scheduler. In a pilot-job model [14–16], generic pilot jobs are submitted to the infrastructure instead of application tasks. When these jobs reach a computing site, they fetch tasks from the pilot manager. Tasks then download input files, execute, and upload their results. To increase reliability and performance, tasks can also be replicated as described in [17]. Task replicas may also be aborted to limit resource waste. This science gateway model totally applies to e-bioinfra, and partly to the P-Grade portal, the Science-Gateway framework in [12], medigrid-DE, and CBRAIN.

Our science-gateway archive model adopts the schema on Fig. 2. **Task** contains information such as final status, exit code, timestamps of internal steps, application and workflow activity name. Each task is associated to a **Pilot Job**. **Workflow Execution** gathers all the activities and tasks of a workflow execution, **Site** connects pilots and tasks to a grid site, and **File** provides the list of

¹ <http://cbrain.mcgill.ca>

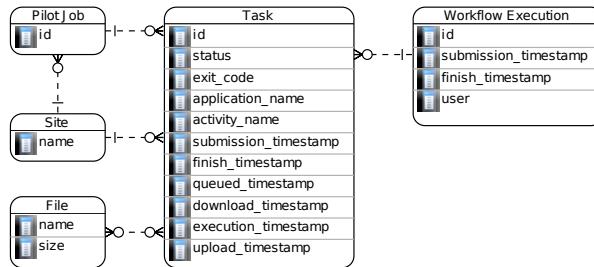


Fig. 2. Science-gateway archive model

files associated to a task and workflow execution. In this work we focus on **Task**, **Workflow Execution** and **Pilot Job**.

The science-gateway archive is extracted from VIP. **Task**, **Site** and **Workflow Execution** information are acquired from databases populated by the workflow engine at runtime. **File** and **Pilot Job** information are extracted from the parsing of task standard output and error files.

Studies presented in the following Sections are based on the workload of the VIP from January 2011 to April 2012. It consists of 2,941 workflow executions, 112 users, 339,545 pilot jobs, 680,988 tasks where 338,989 are completed tasks, 138,480 error tasks, 105,488 aborted tasks, 15,576 aborted task replicas, 48,293 stalled tasks and 34,162 submitted or queued tasks. Stalled tasks are tasks which lost communication with the pilot manager, e.g. because they were killed by computing sites due to quota violation. Tasks ran on the biomed virtual organization of the European Grid Infrastructure (EGI²). Traces used in this work are available to the community in the Grid Observatory³.

3 Case Studies

3.1 Pilot Jobs

Pilot jobs are increasingly used to improve scheduling and reliability on production grids [14–16]. This type of workload, however, is difficult to analyze from infrastructure traces as a single pilot can wrap several tasks, which remains unknown to the infrastructure. In our case, pilots are discarded after 5 task executions, if the remaining walltime allowed on the site cannot be obtained, if they are idle for more than 10 minutes, or if one of their tasks fails. Pilots can execute any task submitted by the science gateway, regardless of the workflow execution and user.

Fig. 3 shows the number of tasks and users per pilot in the archive. Out of the 646,826 executed tasks in the archive, only those for which a standard output containing the pilot id could be retrieved are represented. This corresponds to

² <http://www.egi.eu>

³ <http://www.grid-observatory.org>

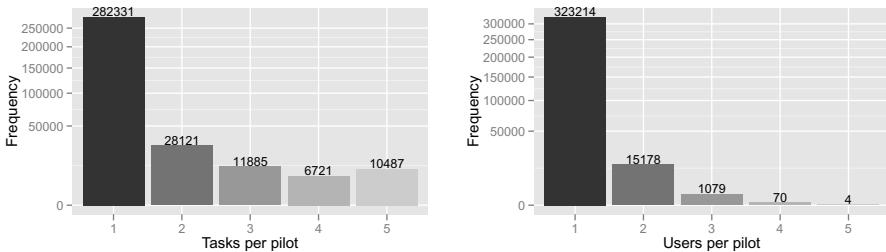


Fig. 3. Histogram of tasks per pilot (left) and users per pilot (right)

453,547 tasks, i.e. 70% of the complete task set. Most pilots (83%) execute only 1 task due to walltime limits or other discards. These 83% execute 282,331 tasks, which represents 62% of the considered tasks. Workload acquired at the infrastructure level would usually assimilate pilot jobs to tasks. Our data shows that this hypothesis is only true for 62% of the tasks. The distribution of users per pilots has a similar decrease: 95% of the pilots execute tasks of a single user.

3.2 Accounting

On a production platform like EGI, accounting data consists of the list of active users and their number of submitted jobs, consumed CPU time, and wall-clock time. Here, we compare data provided by the infrastructure-level accounting services of EGI⁴ to data obtained from the science-gateway archive.

Fig. 4 compares the number of users reported by EGI and the scientific gateway. It shows a dramatic discrepancy between the two sources of information, explained by the use of a robot certificate in the gateway. Robot certificates are regular X.509 user certificates that are used for all grid operations performed by a science gateway, namely data transfers and task submission. From an EGI point of view, all VIP users are accounted as a single user regardless of their real identity. EGI reports more than one user for months 12, 13, 15 and 16 due to updates of the VIP certificate. The adoption of robot certificates totally discards the accounting of user names at the infrastructure level. Studies such as presented on Fig. 17 in [18] or on Fig. 1 in [2], cannot be considered reliable in this context. Robot certificates are not an exception: a survey available online⁵ shows that 80 of such certificates are known on EGI. By avoiding the need for users to request personal certificates, they simplify the access to the infrastructure to a point that their very large adoption in science gateways seems unavoidable.

Fig. 5 compares the number of submitted jobs, consumed CPU time, and consumed wall-clock time obtained by the EGI infrastructure and by VIP. The number of jobs reported by EGI is almost twice as important as in VIP. This huge discrepancy is explained by the fact that many pilot jobs do not register to the pilot system due to some technical issues, or do not execute any task due to the

⁴ <http://accounting.egi.eu>

⁵ https://wiki.egi.eu/wiki/EGI_robot_certificate_users

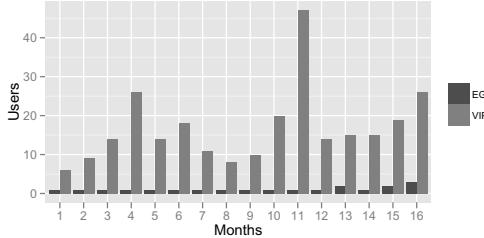


Fig. 4. Number of reported EGI and VIP users

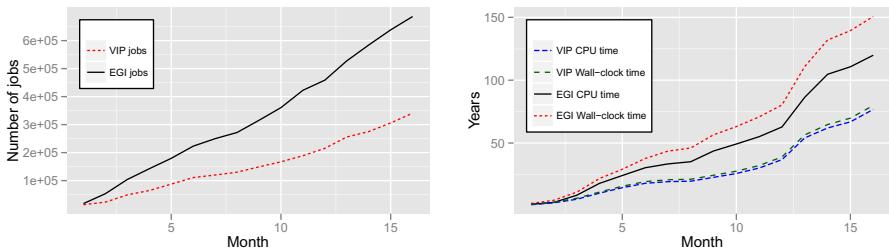


Fig. 5. Number of submitted pilot jobs (*left*), and consumed CPU and wall-clock time (*right*) by the infrastructure (EGI) and the science gateway (VIP)

absence of workload, or execute tasks for which no standard output containing the pilot id could be retrieved. These pilots cannot be identified from the task logs. While this highlights serious potential improvements in the pilot manager, it also reveals that a significant fraction of the workload measured by EGI does not come from applications but from artifacts introduced by pilot managers. This should be taken into account when conducting studies on application-level schedulers from workload acquired at the infrastructure level.

About 60 walltime years are missing from the science gateway archive, compared to the infrastructure. This is due to the pilot setup time (a few minutes per pilot), and to the computing time of lost tasks, for which no standard output containing monitoring data could be retrieved. Tasks are lost (a.k.a stalled) in case of technical issues such as network interruption or deliberate kill from sites due to quota violation. Better investigating this missing information is part of our future work.

3.3 Task Analysis

Traces acquired at the science-gateway level provide fine-grained information about tasks, which is usually not possible at the infrastructure level. Fig. 6 shows the distributions of download, upload and execution times for successfully completed tasks. Distributions show a substantial amount of very long steps.

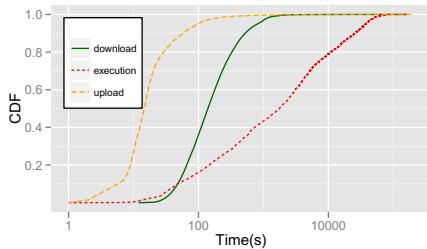


Fig. 6. Different steps in task life

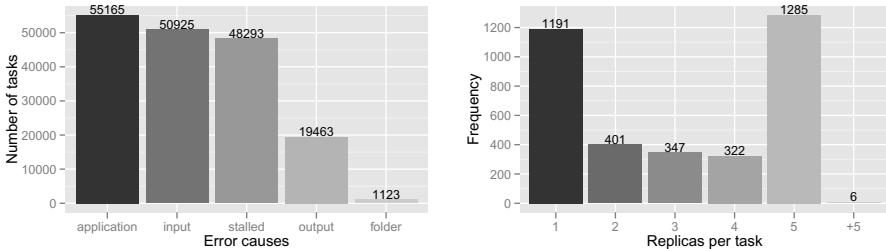


Fig. 7. Task error causes (left) and number of replicas per task (right)

Error causes can also be investigated from science-gateway archives. Fig. 7 (left) shows the occurrence of 6 task-level errors. These error codes are application-specific and not accessible to infrastructure level archives, see e.g. [19] (Table 3) and [3].

A common strategy to cope with recoverable errors is to replicate tasks [20], which is usually not known to the infrastructure. Fig. 7 (right) shows the occurrence of task replication in the science-gateway archive.

3.4 Bag of Tasks

In this section, we evaluate the accuracy of the method presented in [8] to detect bag of tasks (BoT). This method considers that two tasks successively submitted by a user belong to the same BoT if the time interval between their submission times is lower or equal to a time Δ . The value of Δ is set to 120s as described in [8]. Fig. 8 presents the impact of Δ on BoT sizes (a.k.a. batch sizes) for $\Delta = 10s, 30s, 60s$ and $120s$.

Fig. 9 presents the comparison of BoT characteristics obtained from the described method for $\Delta = 120s$ and from VIP. BoTs in VIP were extracted as the tasks generated by the same activity in a workflow execution. Thus, they can be considered as ground truth and are named **Real Non-Batch** for single-task BoTs and **Real Batch** for others. Analogously, we name **Non-Batch** and **Batch** BoTs determined by the method. **Batch** has about 90% of its BoT sizes ranging from 2 to 10 while these batches represent about 50% of **Real Batch**. This discrepancy has a direct impact on the BoT duration (makespan), inter-arrival time and consumed CPU time. The duration of **Non-Batch** are overestimated up to 400%,

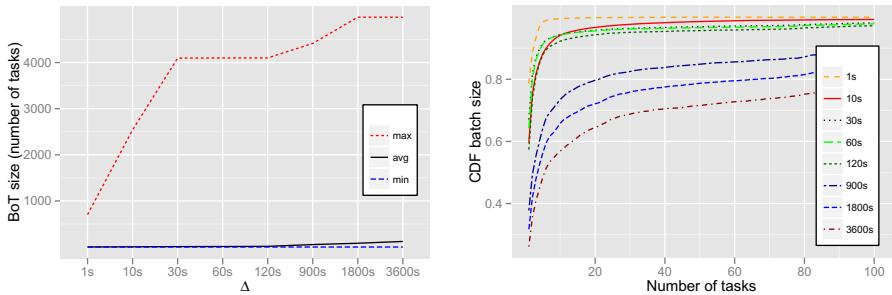


Fig. 8. Impact of parameter Δ on BoT sizes: minimum, maximum and average values (*left*); and its distribution (*right*)

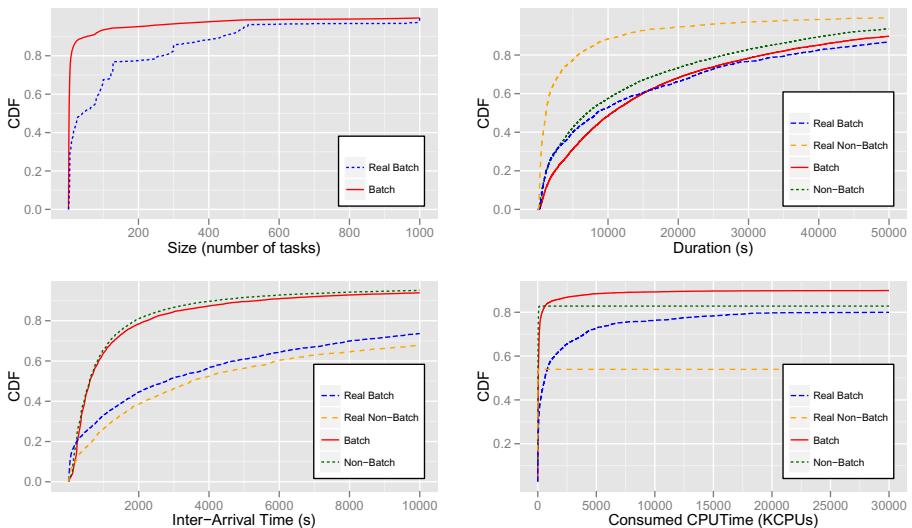


Fig. 9. CDFs of characteristics of batched and non-batched submissions: *BoT sizes*, *duration per BoT*, *inter-arrival time* and *consumed CPU time* ($\Delta = 120s$)

inter-arrival times for both **Batch** and **Non-Batch** are underestimated by about 30% in almost all intervals, and consumed CPU times are underestimated of 25% for **Non-Batch** and of about 20% for **Batch**. This data shows that detecting bag of tasks based on infrastructure-level traces is very inaccurate. Such inaccuracy may have important consequences on works based on such detection, e.g. [21].

3.5 Workflows

Few works study the characterization of grid workflow executions. In [5], the authors present the characterization of 2 workloads that are mostly test runs conducted by developers. To the best of our knowledge, there is no work on the characterization of grid workflows in production.

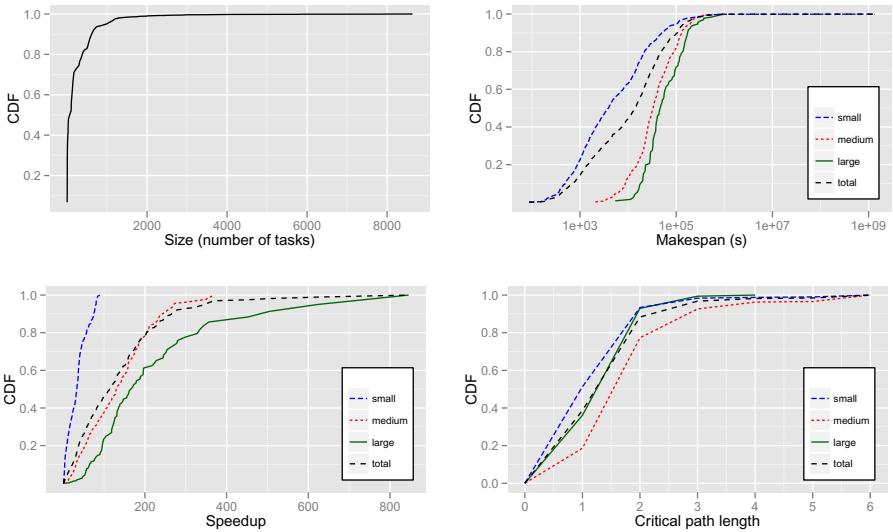


Fig. 10. Characteristics of workflow executions: number of tasks (*top left*), CPU time and makespan (*top right*), speedup (*bottom left*) and critical path length (*bottom right*).

Fig. 10 presents characteristics of the workflow executions extracted from our science-gateway archive. They could be used to build workload generators for the evaluation of scheduling algorithms. Let N be the number of tasks in a workflow execution; we redefine the 3 classes presented in [5] to **small** for $N \leq 100$, **medium** for $100 < N \leq 500$ and **large** for $N > 500$. From Fig. 10 (top left), we observe that the workload is composed by 52%, 31% and 17% of **small**, **medium** and **large** executions respectively. In Fig. 10 (top right), 90% of **small**, 66% of **medium** and 54% of **large** executions have a makespan lower than 14 hours. Speedup values presented in Fig. 10 (bottom left) show that execution speedup increases with the size of the workflow, which indicates good parallelization. Critical path lengths are mostly up to 2 levels for **small** and **large** executions and up to 3 for **medium** executions (bottom right of Fig. 10).

4 Conclusion

We presented a science-gateway model of workload archive containing detailed information about users, pilot jobs, task sub-steps, bag of tasks and workflow executions. We illustrated the added value of science-gateway workloads compared to infrastructure-level traces using information collected by the Virtual Imaging Platform in 2011/2012, which consist of 2,941 workflow executions, 339,545 pilot jobs, 680,988 tasks and 112 users that consumed about 76 CPU years.

Several conclusions demonstrate the added-value of a science-gateway approach to workload archives. First, it can exactly identify tasks and users, while infrastructure-level traces cannot identify 38% of the tasks due to their bundling

in pilot jobs, and cannot properly identify users when robot certificates are used. Infrastructure archives are also hampered by additional workload artifacts coming from pilot-job schedulers, which can be distinguished from application workload using science-gateway archives. More detailed information about tasks is also available from science-gateway traces, such as distributions of download, upload and execution times, and information about replication. Besides, the detection of bag of tasks from infrastructure traces is inaccurate, while a science-gateway contains ground truth. Finally, we reported a few parameters on workflow executions, which could not be extracted from infrastructure-level traces. Limits of science-gateway workloads still exist. In particular, it is very common that a significant fraction of lost tasks do not report complete information.

Traces acquired by the Virtual Imaging Platform will be regularly made available to the community in the Grid Observatory. We hope that other science-gateway providers could also start publishing their traces so that computer-science studies can better investigate production conditions. Information provided by such science-gateway archives can be used, to elaborate benchmarks, to simulate applications and algorithms targeting production systems, or to feed algorithms with historical information [17].

Studies presented in this work only show a partial overview of the potential of science-gateway traces. In particular, information about file access pattern, about the number and location of computing sites used per workflow or bag-of-task execution, and about task resubmission is available in the archive.

Acknowledgment. This work is funded by the French National Agency for Research under grant ANR-09-COSI-03 “VIP”. We thank the European Grid Initiative and National Grid Initiatives, in particular France-Grilles, for the infrastructure and support.

References

1. Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., Epema, D.H.J.: The grid workloads archive. *Future Gener. Comput. Syst.* 24(7), 672–686 (2008)
2. Iosup, A., Epema, D.: Grid computing workloads: bags of tasks, workflows, pilots, and others. *IEEE Internet Computing* 15(2), 19–26 (2011)
3. Kondo, D., Javadi, B., Iosup, A., Epema, D.: The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In: CCGrid 2010, pp. 398–407 (2010)
4. Germain-Renaud, C., Cady, A., Gauron, P., Jouvin, M., Loomis, C., Martyniak, J., Nauroy, J., Philippon, G., Sebag, M.: The grid observatory. In: IEEE International Symposium on Cluster Computing and the Grid, pp. 114–123 (2011)
5. Ostermann, S., Prodan, R., Fahringer, T., Iosup, R., Epema, D.: On the characteristics of grid workflows. In: CoreGRID Symposium - Euro-Par 2008 (2008)
6. Christodoulopoulos, K., Gkamas, V., Varvarigos, E.: Statistical analysis and modeling of jobs in a grid environment. *Journal of Grid Computing* 6, 77–101 (2008)
7. Medernach, E.: Workload Analysis of a Cluster in a Grid Environment. In: Feitelson, D.G., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2005. LNCS, vol. 3834, pp. 36–61. Springer, Heidelberg (2005)

8. Iosup, A., Jan, M., Sonmez, O., Epema, D.: The Characteristics and Performance of Groups of Jobs in Grids. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 382–393. Springer, Heidelberg (2007)
9. Ferreira da Silva, R., Camarasu-Pop, S., Grenier, B., Hamar, V., Manset, D., Montagnat, J., Revillard, J., Balderrama, J.R., Tsaregorodtsev, A., Glatard, T.: Multi-Infrastructure Workflow Execution for Medical Simulation in the Virtual Imaging Platform. In: HealthGrid 2011, Bristol, UK (2011)
10. Shahand, S., Santcroos, M., Mohammed, Y., Korkhov, V., Luyf, A.C., van Kampen, A., Olabarriaga, S.D.: Front-ends to Biomedical Data Analysis on Grids. In: Proceedings of HealthGrid 2011, Bristol, UK (June 2011)
11. Kacsuk, P.: P-GRADE Portal Family for Grid Infrastructures. *Concurrency and Computation: Practice and Experience* 23(3), 235–245 (2011)
12. Ardizzone, V., Barbera, R., Calanducci, A., Fargetta, M., Ingrà, E., La Rocca, G., Monforte, S., Pistagna, F., Rotondo, R., Scardaci, D.: A European framework to build science gateways: architecture and use cases. In: 2011 TeraGrid Conference: Extreme Digital Discovery, pp. 43:1–43:2. ACM, New York (2011)
13. Krefting, D., Bart, J., Beronov, K., Dzhimova, O., Falkner, J., Hartung, M., Hoheisel, A., Knoch, T.A., Lingner, T., Mohammed, Y., Peter, K., Rahm, E., Sax, U., Sommerfeld, D., Steinke, T., Tolxdorff, T., Vossberg, M., Viezens, F., Weisbecker, A.: Medigrid: Towards a user friendly secured grid infrastructure. *Future Generation Computer Systems* 25(3), 326–336 (2009)
14. Luckow, A., Weidner, O., Merzky, A., Maddineni, S., Santcroos, M., Jha, S.: Towards a common model for pilot-jobs. In: HPDC 2012, Delft, The Netherlands (2012)
15. Tsaregorodtsev, A., Brook, N., Ramo, A.C., Charpentier, P., Closier, J., Cowan, G., Diaz, R.G., Lanciotti, E., Mathe, Z., Nandakumar, R., Paterson, S., Romanovsky, V., Santinelli, R., Sapunov, M., Smith, A.C., Miguelz, M.S., Zhelezov, A.: DIRAC3. The New Generation of the LHCb Grid Software. *Journal of Physics: Conference Series* 219(6), 062029 (2009)
16. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience* 17(2-4), 323–356 (2005)
17. Ferreira da Silva, R., Glatard, T., Desprez, F.: Self-healing of operational workflow incidents on distributed computing infrastructures. In: IEEE/ACM CCGrid 2012, Ottawa, Canada, pp. 318–325 (2012)
18. Ilijasic, L., Saitta, L.: Characterization of a Computational Grid as a Complex System. In: Grid Meets Autonomic Computing (GMAC 2009), pp. 9–18 (June 2009)
19. Lingrand, D., Montagnat, J., Martyniak, J., Colling, D.: Optimization of jobs submission on the EGEE production grid: modeling faults using workload. *Journal of Grid Computing (JOGC) Special Issue on EGEE* 8(2), 305–321 (2010)
20. Casanova, H.: On the harmfulness of redundant batch requests. In: International Symposium on High-Performance Distributed Computing, pp. 255–266 (2006)
21. Brasileiro, F., Gaudencio, M., Silva, R., Duarte, A., Carvalho, D., Scardaci, D., Ciuffo, L., Mayo, R., Hoeger, H., Stanton, M., Ramos, R., Barbera, R., Marechal, B., Gavillet, P.: Using a simple prioritisation mechanism to effectively interoperate service and opportunistic grids in the eela-2 e-infrastructure. *Journal of Grid Computing* 9, 241–257 (2011)

Energy Adaptive Mechanism for P2P File Sharing Protocols

Mayank Raj¹, Krishna Kant², and Sajal K. Das¹

¹ Center for Research in Wireless Mobility and Networking (CReWMaN),
Department of Computer Science and Engineering, University of Texas at Arlington
`mayank.raj@mavs.uta.edu, das@uta.edu`
² George Mason University
`kkant@gmu.edu`

Abstract. Peer to peer (P2P) file sharing applications have gained considerable popularity and are quite bandwidth and energy intensive. With the increased usage of P2P applications on mobile devices, its battery life has become of significant concern. In this paper, we propose a novel mechanism for energy adaptation in P2P file sharing protocols to significantly enhance the possibility of a client completing the file download before exhausting its battery. The underlying idea is to group mobile clients based on their energy budget and impose restrictions on bandwidth usage and hence on energy consumption. This allows us to provide favoured treatment to low energy devices, while still ensuring long-term fairness through a credit based mechanism and preventing free riding. Furthermore, we show how the proposed mechanism can be implemented in a popular P2P file sharing application, the BitTorrent protocol and analyze it through a comprehensive set of simulations.

Keywords: P2P file sharing application, Energy Adaptive Computing, BitTorrent, Energy groups.

1 Introduction

As mobile devices become more and more indispensable part of our daily lives, the popular desktop applications, such as the peer to peer (P2P) file sharing applications, will continue to migrate to such devices. However, P2P file sharing applications are typically bandwidth and energy intensive and can be a significant drain on the mobile battery.

In this paper, we propose a novel mechanism for adapting the bandwidth usage of P2P file sharing applications based on the energy constraints of the devices for higher energy efficiency [1]. Specifically, we provide differentiated service to clients based on their energy budget. The battery constrained clients define an energy budget for downloading the file which enables them to adapt their contributions to the network and the service they receive from the network based on it. The protocol ensures the provisioning and delivery of the desired service rate to the clients based on their energy budget. To prove the feasibility of the

proposed mechanism, we discuss the implementation of the proposed mechanism in the context of BitTorrent protocol, a popular P2P file sharing protocol.

The paper is organized as follows. In Section 2 we discuss related works. Section 3 introduces the proposed mechanism and we discuss the related challenges and solutions in the implementation of the proposed mechanism in Section 4. In Section 5, we briefly discuss the implementation of our proposed mechanism in the context of BitTorrent protocol and analyze it through simulation. Section 6 concludes the paper with directions for future work.

2 Related Works

Several solutions have been proposed in the literature to improve the energy efficiency of the P2P file sharing protocols. Middleware based solution as discussed in [2–4], where the task of downloading the file is delegated to a proxy [2, 4] or cloud [5] for higher energy efficiency. The end device only wakes up to receive the file from the proxy or cloud when it has been downloaded. Thus by transferring the computational and protocol overheads to an external device, the mobile devices conserve energy. Different mechanisms for deployment of proxy-based solutions are described in [3]. In [5], a cloud based solution is presented, wherein the BitTorrent client is running remotely in the cloud and is controlled through a thin interface on the mobile device. Though the middleware based approaches guarantee that the file is downloaded irrespective of the battery constraints of the mobile devices, they must exist in a trusted or a controlled environment for security and privacy. Furthermore, the availability of middleware may be local to a network and may not be ubiquitously available. In [6], the task of connection management and query processing of the P2P protocol is delegated to a low cost external hardware based proxy to conserve energy. However, this approach requires the use of additional hardware, which is not desired. In [7], a green BitTorrent protocol is proposed, which introduces a sleep state in the TCP protocol to minimize transmissions. To prevent snubbing of sleeping peers, the authors presented an architecture as well as a mechanism to distinguish between sleeping and dead peers.

Unlike the discussed approaches, we address the problem of adapting the bandwidth usage of the P2P file sharing clients based on their energy budget for higher energy efficiency. We define such approaches under the ambit of Energy Adaptive Computing (EAC) [1]. Since the proposed mechanism requires modifications only to the application protocol, it can be used in conjunction with any traditional energy conservation techniques. To the best of our knowledge, no previous approaches in the literature aim at empowering the P2P file sharing applications with EAC.

3 Energy Adaptive Mechanism for P2P File Sharing Applications

The proposed mechanism exploits the following two characteristic of the mobile devices to adapt their energy consumption based on their energy budgets.

1) Mobile device consumes more energy when transmitting than receiving. Essentially, during transmission, the signal is amplified to achieve the desired signal to noise ratio for successful decoding at the receiver. Thus attributing for the higher energy consumption. 2) It is much more energy efficient for the mobile device to download the file faster, i.e. at high download rate. After each transmission or reception, mobile devices continue to remain in active state for a short duration, called tail time, in anticipation of another packet. Frequent occurrence of tail time can result in significant energy consumption for the mobile devices [8]. At high download rates, packets are either received in the tail time or in large single bursts, thus, preventing the frequent occurrence of tail time and reducing the average energy per transfer [8]. In the following sections we discuss the network scenario and show how the above mentioned energy consumption characteristics are exploited in the proposed mechanism.

3.1 Network Scenario

In this paper, we consider a content sharing network, wherein the content is shared and distributed among users using a partially decentralized P2P file sharing protocol. Each content can either be free or paid, wherein paid content are purchased using credits. Credits can be purchased or gained through participation in the network, as explained in Section 3.2. For accounting and credit management, each user is assigned a unique id, using which it must authenticate itself before joining the network. We refer each user in the network as peer.

In partially decentralized P2P file sharing applications, a super node or central server monitors and maintains information about the peers and the network. Peers prior to joining the network must communicate with it, to be able to identify the neighboring peers. We define neighboring peers of a peer P , as those which are downloading the same file as P . Each file being downloaded is divided into smaller pieces of fixed size. Peers exchange or download pieces of mutual interest from each other. After a peer has received all the pieces of the file, it combines them to reconstruct the whole file. On completion of download, a peer may then choose to stay back in the network and keep providing pieces of file to other peers as a “seed”.

The network is assumed to be composed of both battery constrained peers, like smartphones and tablets, and non battery constrained peers, like desktops. Each battery constrained peer defines an energy budget for downloading the file prior to joining the network. Energy budget of a battery constrained peer is the maximum amount of energy it wants to consume for downloading the file. It is determined while taking into consideration the available energy of the device, the energy consumption and bandwidth usage profile of other applications running on it and the maximum bandwidth the device can allocate to the application for downloading the file.

3.2 Description of the Proposed Mechanism

In the proposed mechanism, peers are divided into two groups, namely Energy Sufficient (ES) group and Energy Constrained (EC) group. Peers in ES group are characterized as either those who are not battery constrained or battery constrained peers who can successfully download the file within their specified energy budget. Peers in ES group download the file only from neighboring peers in the same group as themselves. No restrictions are imposed on the bandwidth usage of peers in ES group. A battery constrained peer, on joining the network can determine whether it will be able to download the file successfully within its given energy budget based on the average upload and download rate of the peers in the ES group. If not, it becomes part of the EC group. The energy consumption of the battery constrained peer for a given bandwidth usage profile can be determined using the Stochastic KiBaM model [9]. The model is fast and reliable with a maximum error of 2.65%, hence, is suitable for real-time applications. Thus peers in EC groups are characterized as comprising of battery constrained peers who cannot download the file within their specified energy budget.

To facilitate peers in EC group to download the file within their energy budget, we divide the peers into smaller energy groups. This subdivision enables us to group peers with similar energy budget together and provide differentiated service to them based on it. The differentiated service allows the peers to adapt their bandwidth usage and download the file. Each energy group is characterized as having a unique energy consumption profile depending on the maximum upload and download rate to which peers in that group must adhere to while downloading the file. Specifically, we exploit the fact that it is much more energy efficient for the mobile devices to operate at high download and upload rates. Thus groups with high permissible upload and download rate can be characterized as having low energy consumption. The energy consumption in the group increases as the maximum permissible upload and download rate of the group decreases. To download a file within its energy budget, peers join an energy group with lower energy consumption profile than its energy budget.

Moreover, we impose the following restrictions on the communication pattern of peers. Peers in an energy group can only download the file from neighboring peers in the same group as themselves, using the traditional P2P file sharing protocol. We assume peers in each energy group upload at the maximum permissible upload rate of the group. In P2P file sharing protocol, peers prefer to associate with neighboring peers having same or higher upload rates than them. Since the P2P file sharing application works on the principle of mutual exchange, the restrictions on communication pattern of the peers allows them to discover neighboring peers with similar upload rates, thus, allowing the network to converge to a stable state faster [10].

In P2P file sharing application, the download rate the peer gets is proportional to the its upload rate. Since no peer will upload more data to a neighboring peer than what it has downloaded from it, we can say the average degree of proportionality is 1, assuming the contributions from seeds are insignificant

when compared with those of neighboring peers. Thus the average download rate the peer gets in an energy group is equivalent to the maximum upload rate of the group. The peer can also get additional download rates from seeds. However, the download rate of the peer may still be not sufficient for it to download the file within its specified energy budget. Hence, we allow the peers in energy groups to receive additional download rates from neighboring peers in ES group. We assume the peers in ES have some residual energy or are not battery constrained, and hence, can provide additional download rates to neighboring peers in energy groups in exchange of credits.

The use of credits for purchasing additional download rates from neighboring peers in ES group has the following advantages. 1) It prevents free riding. 2) In traditional P2P file sharing applications, a peer must contribute to the neighboring peers in ES group to increase its download rate. The use of credits allows peers to gain additional download rate at low energy cost as more energy is consumed when transmitting than receiving. 3) It provides long-term fairness. Peers can accumulate credits when downloading a file by providing additional download rates to neighboring peers in other groups. The credits can be used by battery constrained peers in future to enhance their download rate and lower their energy consumption when they join the network to download a file with low energy budget. Furthermore, both non battery-constrained and battery-constrained peers, can use the gained credits to purchase content in the network. The rate of exchange of credits and bandwidth is kept high for low energy group and it decreases from low to high energy. Low energy group peers are given more preference when allocating additional download rates as they are operating at low and strict energy budgets. Such a variable exchange rate promotes fairness as it compensates for the favourable treatment. Furthermore, it dissuades peers from joining energy group with high download rates (low energy groups) owing to their higher exchange rate. Figure 1 shows the interactions between peers in three energy groups, namely low, medium and high, and ES group, based on the proposed mechanism.

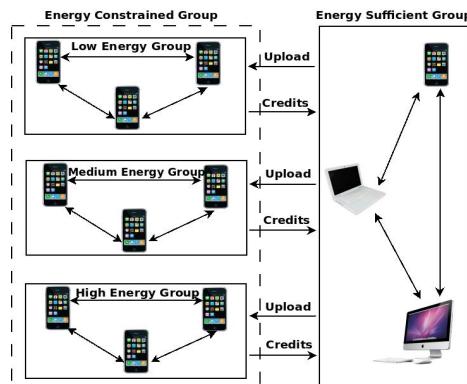


Fig. 1. Interaction between peers in and across groups

4 Challenges and Solutions in Implementation of the Proposed Mechanism

We discuss some of the challenges in the implementation of the proposed mechanism and propose solutions to them. The proposed mechanism requires us to address the following challenges; 1) design and creation of energy groups; and 2) allocation of the additional download rates to peers in energy groups based on demands.

4.1 Design and Creation of Energy Groups

Let E be the set of energy groups, where $|E| = M$. For each energy group $E_i \in E$, let U_{E_i} and D_{E_i} be the maximum upload and download rate of the group. The set of energy groups E are ordered by the maximum upload rate of each group in descending order, i.e. $U_{E_1} > U_{E_2} > \dots > U_{E_M}$. Since a peer can get additional download rates from seeds as well as purchase from neighboring peers in the ES group, the maximum download rate it can receive in an energy group is kept higher than the maximum upload rate of that group, i.e. $D_{E_i} > U_{E_i}$. To keep the energy consumption profile of each group unique, the maximum download rate of each group of each group is kept as $D_{E_1} > D_{E_2} > \dots > D_{E_M}$. The minimum energy with which a peer can download a file in an energy group is given as when the peer is performing at the maximum upload and download rate of the group. Since, $(U_{E_1} + D_{E_1}) > (U_{E_2} + D_{E_2}) > \dots > (U_{E_M} + D_{E_M})$, the minimum energy consumption in the energy groups can be given as $E_1 < E_2 < \dots < E_M$. Thus group E_1 has the minimum energy consumption and requires peers to dedicate the highest amount of bandwidth for downloading the file. The minimum energy consumption of the groups increases as we decrease the maximum permissible upload rate of the peers in the group.

However, we must first be able to restrict the upload and download rates of peers in P2P file sharing applications to be able to create energy groups. Hence we carried out a survey of mobile P2P file sharing applications to determine its feasibility. It was revealed that mobile applications, like Transmission P2P App for Nokia N900 and MobileMule, of some popular P2P file sharing protocols, like BitTorrent [11] and eMule [12] respectively, allow us to specify restrictions on the maximum upload and download rate that can be used by them. Thus establishing the feasibility of limiting the bandwidth usage of peers for creation of energy groups.

4.2 Distribution of Additional Download Rate to Peers

On joining ES group, each peer announces the amount of additional upload rate it wishes to offer. Similarly, on joining an energy group each peer announces the amount of additional download rate it wishes to purchase from neighboring peers in ES group. The central server must provide peers in ES group wishing to offer additional download rates with a set of neighboring peers in energy groups

who wish to purchase it and inform how to distribute it among them. The total additional download rate (V_{E_i}) being requested by peers in energy group E_i is given as $V_{E_i} = w_{E_i} \cdot \min(R_{E_i}, BW_{E_i})$. Where R_{E_i} is the total download rate the peers in group E_i wish to purchase and w_i is the weight assigned to group E_i , such that $w_{E_1} > w_{E_2} > \dots > w_{E_M}$. This ensures that the peers with low energy budget get higher preference when allocating additional download rates to them. BW_{E_i} is the total average download rate required by peers in group E_i to make their download rate equal to maximum download rate D_{E_i} of the group, as given in Equation 1.

$$BW_{E_i} = \begin{cases} 0 & \text{if } D_{E_i} \leq \bar{\delta}_{E_i} \\ (D_{E_i} - \bar{\delta}_{E_i}) \cdot \bar{x}_{E_i} & \text{if } D_{E_i} > \bar{\delta}_{E_i} \end{cases} \quad (1)$$

In Equation 1, \bar{x}_{E_i} represents the average number of peers in group E_i and δ_{E_i} is the average download rate of the peers in group E_i . Thus V_{E_i} gives the total weighted download rate that peers in energy group E_i can purchase or want to purchase. The total additional download rate being offered by a peer in ES group is distributed among the energy groups in the ratio of their demands, i.e. $(\frac{V_{E_1}}{\sum_{E_i \in E} V_{E_i}}, \frac{V_{E_2}}{\sum_{E_i \in E} V_{E_i}}, \dots, \frac{V_{E_M}}{\sum_{E_i \in E} V_{E_i}})$. The total additional download rate allocated to energy group E_i is distributed among the peers in that group in the ratio of their individual demands.

5 Implementation and Simulation of the Proposed Mechanism

To evaluate the proposed mechanism, we first discuss its implementation in the context of the BitTorrent protocol [11], a popular partially decentralized P2P file sharing application, and analyze its performance using simulation.

5.1 The BitTorrent Protocol

The BitTorrent [11] protocol employs a centralized tracker mechanism for peer discovery. A peer, say P , intending to download a file, say F , first downloads the “.torrent” file containing information about the tracker and file F from the host web-server. The tracker maintains a list of peers currently downloading the file F . Upon contacting the tracker, peer P receives a list of neighboring peers that it may contact for exchanging chunks of the file F . The peer P individually contacts the neighboring peers and requests them to exchange chunks of file with itself. On mutual consent, the peers exchange chunks of the file F of mutual interest.

The tracker, in the BitTorrent protocol, assumes the function of central server. The tracker maintains the group membership information of each peer and their choice of action, i.e. whether they wants to provide additional download rate or purchase additional download rate, along with their identity. Each peer is provided with two sets of neighboring peers. The first set consists of neighboring

peers in the same energy group as themselves from which they can download the file using the BitTorrent protocol. For peers in energy groups, the second set contains neighboring peers in ES from which it can purchase additional download rates. For peer in ES group, the second set contains neighboring peers in energy groups to which it should provide additional download rates. The tracker runs the allocation algorithm for allocating additional download rates and informs the peers in ES group how to distribute it among the neighboring peers in the second set.

5.2 Simulation and Results

Furthermore, we carried out simulations to evaluate the performance of the proposed mechanism. Simulations were conducted in ns2 [13] using a BitTorrent module [14] modified for our purpose. We assumed 3 energy groups in EC group, $E = \{low, medium, high\}$ exist, representing peers with low, medium and high energy budget respectively. The maximum upload rate of low, medium and high energy group was considered as 64, 128 and 192 kbps respectively and the maximum download rate as 128, 192 and 256 kbps respectively. The upload rate of the peers in ES group was considered as 256 kbps. We considered a file size of 700 MB. Based on analysis of traces from live BitTorrent P2P network [15], the peer arrival rate for each energy group was kept as 0.0045 peers/sec. We assume a peer on completion of download becomes a seed with probability 0.5. All peers in a group operate at the maximum upload rate of the group as well as the additional download rate peers in a group wish to purchase is equal to the rate required to make their download rate equal to the maximum download rate of the group. The weights assigned to each group was kept as $w_1 : w_2 : w_3 = 3 : 2 : 1$ and the simulation time was kept constant as 4.5 hrs. Each data point represents the average value computed over 10 iterations of each simulation. In the first experiment, we observed the average download rate of peers in each group with varying availability of additional download rate from neighboring peers in ES group, as shown in Figure 2. The additional download rate offered by peers

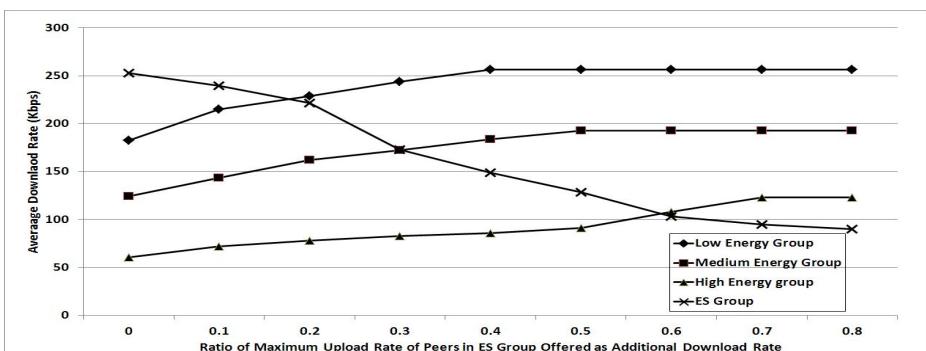


Fig. 2. Average Download Rate of Peers with Varying Availability of Additional Download Rate

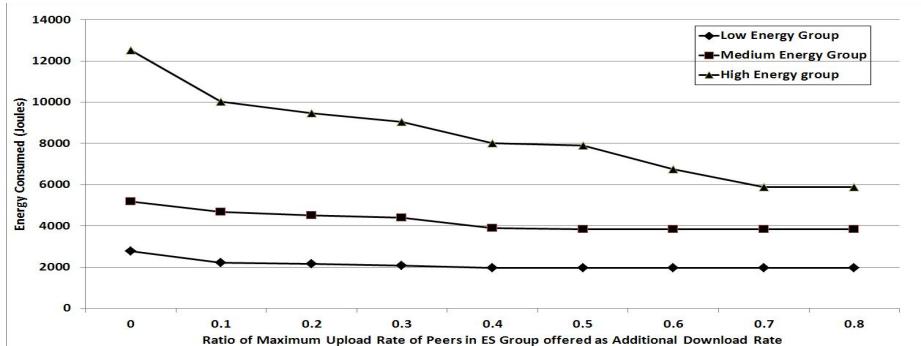


Fig. 3. Average Energy consumption of Peers with Varying Availability of Additional Download Rate

in ES group was varied as the ratio of their maximum upload rate. The allocation algorithm gives higher preference to low energy group as peers in the group achieve maximum download rate faster. The rate at which peers in an energy group approach the maximum download rate of the group is dependent on the weights assigned to each group. When peers in low energy group achieve the maximum download rate of 256 kbps, the peers in medium and high energy group are allocated the remaining additional download rate in the ratio of their demands. Hence, the rate at which peers in medium and high energy group approach the maximum download rate increases. As the peers in the ES group provide more additional download rate, they reduce their contributions to neighboring peers in their group. Since, BitTorrent works on tit for tat principle, peers in ES group receive proportional download rates from neighboring peers in their group and become increasingly more reliable on the the download rate being provided by the seeds, thus, their download rate reduces. Furthermore, using the application traces collected in the first experiment, we generated traffic to (from) Google Nexus One smartphone and measured the energy consumption for downloading the file, as shown in Figure 3. The results establish our claim that peers in low energy groups are able to download the file using less energy than peers in high energy group. The energy consumption increases from medium to high energy group. Thus establishing the feasibility of using energy groups having unique energy consumption profiles to download the file within the specified energy budgets of peers.

In the second experiment, we carried out a study of the effect of arrival rate of peers on the average download rate of each energy group, as shown in figure 4. We kept the fraction of maximum upload rate peers in ES group offer as additional download rate constant at 0.5. The download rate a peer gets in a BitTorrent system is independent of the peer arrival rate [10]. Since, the implementation of the proposed mechanism does not require changes to the inherent characteristics of the BitTorrent protocol but only requires us to put constraints on their bandwidth usage, we inherit the properties of the BitTorrent protocol.

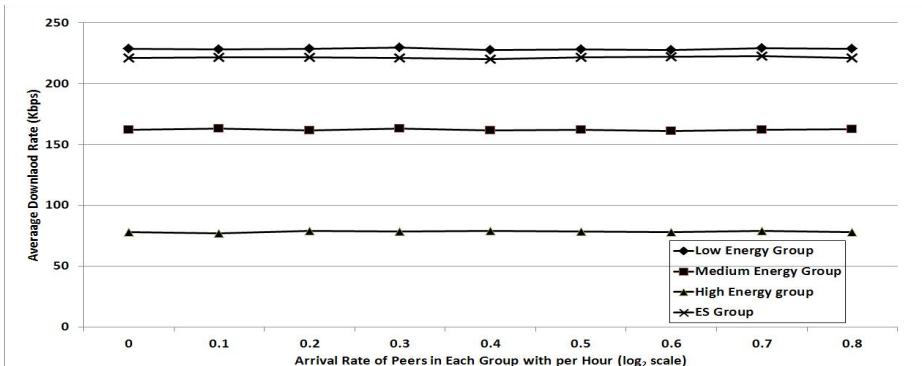


Fig. 4. Average Download Rate of Peers with Varying Arrival Rate

The BitTorrent protocol along with the additional upload rate allocation model ensures that the system remains robust to the varying peer arrival rates.

6 Conclusions

In this paper, we have proposed a framework that effectively allows low energy clients to “borrow” energy from clients with sufficient energy, and thereby increase their probability of downloading the file. We provided a detailed description of challenges and associated solution in the implementation of the proposed framework in the context of BitTorrent protocol. Through an exhaustive set of simulations we analyzed the proposed framework and showed how it can provide differentiated service to low energy clients. Having shown the feasibility and effectiveness of the proposed mechanism, we intend to extend it to other popular P2P file sharing applications, like eMule and BitTorrent DHT protocol, and carry out real-world experiments. Finally, we wish to examine the new security issues in BitTorrent protocol brought in by the credit based mechanism and the proposed energy adaptation capability as well as provide a optimal energy group selection algorithm for battery constrained peers.

References

1. Kant, K., Murugan, M., Du, D.H.C.: Willow: A control system for energy and thermal adaptive computing. In: Proc. of IPDPS (2011)
2. Anastasi, G., Giannetti, I., Passarella, A.: A bittorrent proxy for green internet file sharing: Design and experimental evaluation. Computer Communications 33(7), 794–802 (2010)
3. Ludanyi, A., Nurminen, J., Kelényi, I.: Bittorrent on mobile phones - energy efficiency of a distributed proxy solution. In: Green Computing Conf., pp. 451–458 (2010)
4. Kelényi, I., Nurminen, J.K., Ludányi, A., Lukovszki, T.: Modeling resource constrained bittorrent proxies for energy efficient mobile content sharing. Peer-to-Peer Networking and Applications 5, 163–177 (2012)

5. Kelényi, I., Nurminen, J.K.: Cloudtorrent - energy-efficient bittorrent content sharing for mobile devices via cloud services. In: Proc. of the CCNC, pp. 646–647 (2010)
6. Jimeno, M., Christensen, K.: A prototype power management proxy for gnutella peer-to-peer file sharing. In: Proc. of LCN, pp. 210–212 (2007)
7. Blackburn, J., Christensen, K.: A simulation study of a new green bittorrent. In: ICC 2009: Communications Workshops, pp. 1–6 (2009)
8. Sharma, A., Navda, V., Ramjee, R., Padmanabhan, V.N., Belding, E.M.: Cooltether: energy efficient on-the-fly wifi hot-spots using mobile phones. In: Proc. of CoNEXT, pp. 109–120 (2009)
9. Panigrahi, D., Chiasserini, C., Dey, S., Rao, R., Raghunathan, A., Lahiri, K.: Battery life estimation of mobile embedded systems. In: Proc. of Inrl Conf. on VLSI Design, pp. 57–63 (2001)
10. Qiu, D., Srikant, R.: Modeling and performance analysis of bittorrent-like peer-to-peer networks. In: SIGCOMM 2004: Proc. of the Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 367–378 (2004)
11. Cohen, B.: Incentives build robustness in bittorrent. In: Proc. First Workshop on Economics of Peer-to-peer Systems, Berkely (2003)
12. emule project, <http://www.emule-project.net/>
13. Network simulator (ns2), <http://isi.edu/nsnam/ns/>
14. Hoßfeld, T., Eger, K., Binzenhöfer, A., Kunzmann, G.: Efficient simulation of large-scale p2p networks: packet-level vs. flow-level simulations. In: Proc. of the Second Workshop on Use of P2P, GRID and Agents for the Development of Content Networks, pp. 9–16 (2007)
15. Zhang, B., Iosop, A., Pouwelse, J., Garbacki, P.: The peer-to-peer trace archive: Design and comparative trace analysis. Delft University of Technology, Tech. Rep. PDS-2010-003 (2010)

Tenth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar'2012)

Rosa M. Badia

¹ Barcelona Supercomputing Center, Spain
rosa.m.badia@bsc.es

² Artificial Intelligence Research Institute (IIIA),
Spanish National Research Council (CSIC), Spain

The International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar'2012) was held, on its tenth edition, in Rhodes Island, Greece. For the fourth time, this workshop was organized in conjunction with the Euro-Par annual series of international conferences dedicated to the promotion and advancement of all aspects of parallel computing.

Heterogeneity is emerging as one of the most profound and challenging characteristics of today's parallel environments. From the macro level, where networks of distributed computers, composed by diverse node architectures, are interconnected with potentially heterogeneous networks, to the micro level, where deeper memory hierarchies and various accelerator architectures are increasingly common, the impact of heterogeneity on all computing tasks is increasing rapidly. Traditional parallel algorithms, programming environments and tools, designed for legacy homogeneous multiprocessors, can at best achieve on a small fraction of the efficiency and potential performance we should expect from parallel computing in tomorrow's highly diversified and mixed environments. New ideas, innovative algorithms, and specialized programming environments and tools are needed to efficiently use these new and multifarious parallel architectures. The workshop is intended to be a forum for researchers working on algorithms, programming languages, tools, and theoretical models aimed at efficiently solving problems on heterogeneous platforms.

The topics to be covered include but were not limited to:

- Heterogeneous parallel programming paradigms and models;
- Languages, libraries, and interfaces for different heterogeneous parallel programming models;
- Performance models and their integration into the design of efficient parallel algorithms for heterogeneous platforms;
- Parallel algorithms for heterogeneous or hierarchical systems, including manycores and hardware accelerators (FPGAs, GPUs, etc.);
- Parallel algorithms for efficient problem solving on heterogeneous platforms (numerical linear algebra, nonlinear systems, fast transforms, computational biology, data mining, multimedia, etc.);
- Software engineering for heterogeneous parallel systems;

- Applications on heterogeneous platforms;
- Integration of parallel and distributed computing on heterogeneous platforms;
- Experience of porting parallel software from supercomputers to heterogeneous platforms;
- Fault tolerance of parallel computations on heterogeneous platforms;
- Algorithms, models and tools for grid, desktop grid, cloud, and green computing.

This year edition the workshop attracted a large number of papers, with a total of 28 papers being submitted for consideration. From these 28 papers, 10 were selected by the program committee, with most papers receiving 4 reviews, and only a few of them with 3 reviews.

Additionally to the regular papers, a keynote presentation was given by Enrique Quintana, Universitat Jaume I, SPAIN, about how model order reduction, an important control theory application, which required the use of a moderate size cluster even for a moderate dynamical system only a few years ago, can nowadays be easily solved using an optimized algorithm for a hybrid CPU-GPU platform. A paper about the contents of the talk is also included in these proceedings.

The papers accepted for presentation in the workshop deal about different aspect of heterogeneous computing, such as: new programming models and their extensions for these type of architectures, new scheduling policies for heterogeneous platforms, new methods to solve algorithms, experiences with applications, alternatives for communication algorithms, etc.

Between all the accepted papers, the program committee decided to nominate as best paper the one entitled: *Weighted Block-Asynchronous Iteration on GPU-Accelerated Systems*. However, this was a difficult task since all papers were of high quality and together with the keynote presentation the workshop become an excellent event.

Unleashing CPU-GPU Acceleration for Control Theory Applications

Peter Benner¹, Pablo Ezzatti², Enrique S. Quintana-Ortí³,
and Alfredo Remón³

¹ Max Planck Institute for Dynamics of Complex Technical Systems,
D-39106 Magdeburg, Germany
benner@mpi-magdeburg.mpg.de

² Instituto de Computación, Universidad de la República,
11.300-Montevideo, Uruguay
pezzatti@fing.edu.uy

³ Dpto. de Ingeniería y Ciencia de Computadores,
Universidad Jaume I, 12.071-Castellón, Spain
{quintana,remon}@icc.uji.es

Abstract. In this paper we review the effect of two high-performance techniques for the solution of matrix equations arising in control theory applications on CPU-GPU platforms, in particular advanced optimization via look-ahead and iterative refinement. Our experimental evaluation on the last GPU-generation from NVIDIA, “Kepler”, shows the slight advantage of matrix inversion via Gauss-Jordan elimination, when combined with look-ahead, over the traditional LU-based procedure, as well as the clear benefits of using mixed precision and iterative refinement for the solution of Lyapunov equations.

Keywords: Control theory, matrix equations, matrix inversion, multi-core processors, graphics processors.

1 Introduction

Consider a dynamic linear time-invariant (LTI) system represented, in the state-space model, as

$$\begin{aligned}\dot{x}(t) &= Fx(t) + Bu(t), & t > 0, & x(0) = x^0, \\ y(t) &= Cx(t) + Du(t), & t \geq 0,\end{aligned}\tag{1}$$

where $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$ and $y(t) \in \mathbb{R}^m$ contain, respectively, the states, inputs and outputs of the system, while $x^0 \in \mathbb{R}^n$ stands for its initial state. Here, $F \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $C \in \mathbb{R}^{p \times n}$, $D \in \mathbb{R}^{p \times m}$, n is referred to as the order of the system and, usually, the number of inputs and outputs satisfy $m, p \ll n$. Two important control theory applications are *model order reduction* (MOR) and *linear-quadratic optimal control* (LQOC). In the first one, the goal is to find an alternative dynamic LTI system, of order $r \ll n$, which can accurately replace the original system (1) in subsequent operations [2]. On the other hand, the objective

of LQOC is to determine an “optimal” feedback control law $u(t) = -Kx(t)$, $t \geq 0$, with $K \in \mathbb{R}^{m \times n}$, that stabilizes (1) (i.e., so that all the eigenvalues of $F - BK$ have negative imaginary part) [14].

Large-scale dynamic LTI systems ($n \geq 5,000$) often arise when modeling controlled physical processes by means of partial differential equations [1,2,9]. Reliable numerical methods for MOR and LQOC problems require the solution of certain linear and quadratic matrix equations featuring a high computational cost. For instance, solving the Lyapunov equations for the controllability/observability Gramians associated with (1), via the matrix sign function, roughly requires $2n^3$ floating-point arithmetic operations (or flops) per iteration, with the number of iterations required for convergence varying between 3–4 to a few dozens [15]. Given the theoretical peak performance of current CPU cores (4 double-precision flops/cycle), we can thus estimate that, under perfect conditions (i.e., operating at peak performance for the full execution of the solver), performing one iteration of the matrix sign function on a single CPU core, for an equation of order $n = 10^4$, can cost slightly more than 4 minutes; if the order of the equation grows to $n = 10^5$, the execution time of a single iteration is longer than 69 hours!

In the past [7,8], we have shown how the use of message-passing Lyapunov solvers based on the matrix sign function (and kernels from the distributed-memory linear algebra library ScaLAPACK [10]) provides an appropriate means to solve MOR and LQOC problems of moderate scale ($n \approx 5,000\text{--}10,000$) on a 32-node cluster. Following the trend of adopting graphics processors (GPUs) as hardware accelerators for compute-intensive applications, more recently we have developed hybrid CPU-GPU codes for the solution of these control applications. For instance, we have evaluated the performance of a basic building kernel like the matrix inverse on a platform consisting of a general-purpose multicore from Intel and a “Fermi” GPU from NVIDIA [5].

In this paper we review two advanced optimization techniques for the solution of Lyapunov equations via the matrix sign function on CPU-GPU platforms: 1) the use of look-ahead in the framework of computing the matrix inverse; and 2) the combined use of mixed precision and iterative refinement (MPIR) to accelerate the solution of Lyapunov matrix equations. While a significant part of the theoretical aspects underlying this work has already been exposed in previous work (see, e.g., [5,6]), the principal motivation for revisiting these techniques is the latest evolution of GPU architectures, specifically, the new “Kepler” GPU from NVIDIA, featuring an important increase in the number of cores w.r.t. the previous generation (Fermi), and also a very different ratio of single-to-double precision arithmetic performance.

The rest of the paper is organized as follows. In Section 2 we briefly review the use of the matrix sign function to solve the matrix equations arising in MOR and LQOC problems. In Section 3 we expose how to efficiently combine look-ahead with several basic optimization techniques for CPU-GPU platforms, in the context of matrix inversion. There, we also illustrate the impact of these techniques on the performance of our hybrid CPU-GPU algorithms on a platform equipped

with a Kepler. In Section 4 we revisit a MPIR procedure in combination with the sign function-based solver for the Lyapunov equation, and experimentally evaluate its performance on the target platform. Finally, we complete the paper with some remarks in Section 5.

2 Solving Matrix Equations via the Matrix Sign Function

Balanced truncation (BT) is an efficient absolute-error method for MOR of large-scale LTI systems [2]. The crucial operation when applying BT to (1) is the solution of the dual Lyapunov equations

$$FW_c + W_c F^T + BB^T = 0, \quad F^T W_o + W_o F + C^T C = 0, \quad (2)$$

for the Cholesky (or, alternatively, low-rank) factors of the symmetric positive semi-definite (s.p.d.) Gramians $W_c, W_o \in \mathbb{R}^{n \times n}$. On the other hand, given a pair of weight matrices $R \in \mathbb{R}^{m \times m}$ and $Q \in \mathbb{R}^{p \times p}$, with R s.p.d. and Q symmetric, under certain conditions the optimal control law for the LQOC problem is given by $u(t) = -Kx(t) = -R^{-1}B^T X x(t)$, with $X \in \mathbb{R}^{n \times n}$ being the s.p.d. solution of the algebraic Riccati equation (ARE):

$$F^T X + XF - XBR^{-1}B^T X + C^T QC = 0. \quad (3)$$

Given a matrix $A \in \mathbb{R}^{q \times q}$, the Newton iteration for the matrix sign function is defined as

$$A_0 := A, \quad A_{j+1} := \frac{1}{2}(A_j + A_j^{-1}), \quad j = 0, 1, \dots \quad (4)$$

Provided A has no eigenvalues on the imaginary axis, $\lim_{j \rightarrow \infty} A_j = \text{sign}(A)$ with an asymptotically quadratic convergence rate. Both the solution of the Lyapunov equations in (2) and the ARE in (3) can be obtained via specialized variants of (4). In both cases, the key operation from the point of view of the cost is the computation of the inverse of an $n \times n$ (Lyapunov) or $2n \times 2n$ (ARE) matrix; see [15] for details. Computing the inverse of a matrix costs $2n^3$ flops, if the matrix has no special structure, or just n^3 flops, in case symmetry can be exploited. In general, the inverse of a sparse, banded or tridiagonal matrix is dense and, therefore, these special structures cannot be leveraged to reduce the cost of computing the matrix inverse during the sign function iteration.

3 Efficient Matrix Inversion on CPU-GPU Platforms

While there exist different approaches for the inversion of (general, symmetric and s.p.d.) matrices, in past work [5] we have shown the superior performance of Gauss-Jordan elimination (GJE) over the conventional LU-based matrix inversion when the target is a heterogeneous platform that combines a traditional CPU with a GPU. The reason for the advantage of matrix inversion via GJE is

twofold. First, the procedure performs a constant number of flops per step, facilitating a balanced distribution of the computation between the CPU cores and the GPU. Second, GJE is richer in large matrix-matrix multiplies, an operation that delivers a high FLOPS (flops/sec.) ratio on both CPUs and GPUs.

Figure 1 shows a blocked procedure for the inversion of a general matrix via GJE (right) and the unblocked variant upon which it is built (left). For simplicity, row permutations are not included in our following discussion, though in practice all our GJE-inversion algorithms employ partial pivoting to ensure practical stability of the procedure.

Algorithm: $A := \text{GJE_UNB}(A)$	Algorithm: $A := \text{GJE_BLK}(A)$
Partition $A \rightarrow \begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix}$	Partition $A \rightarrow \begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix}$
where A_{TL} is 0×0	where A_{TL} is 0×0
while $m(A_{TL}) < m(A)$ do	while $m(A_{TL}) < m(A)$ do
Repartition	Determine block size b
$\begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{pmatrix}$	$\begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{pmatrix}$
where α_{11} is a scalar	where A_{11} is $b \times b$
<hr/>	<hr/>
% Column factorization	% Panel factorization
$a_{01} := -a_{01}/\alpha_{11}$	$\begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} := \text{GJE_UNB} \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix}$
$\alpha := \alpha_{11}$	
$a_{21} := -a_{21}/\alpha_{11}$	
% Left update	% Left update
$A_{00} := A_{00} + a_{01}a_{10}^T$	$A_{00} := A_{00} + A_{01}A_{10}$
$A_{20} := A_{20} + a_{21}a_{10}^T$	$A_{10} := A_{11}A_{10}$
$a_{10}^T := a_{10}^T/\alpha$	$A_{20} := A_{20} + A_{21}A_{10}$
% Right update	% Right update
$A_{02} := A_{02} + a_{01}a_{12}^T$	$A_{02} := A_{02} + A_{01}A_{12}$
$A_{22} := A_{22} + a_{21}a_{12}^T$	$A_{12} := A_{11}A_{12}$
$a_{12}^T := a_{12}^T/\alpha$	$A_{22} := A_{22} + A_{21}A_{12}$
$\alpha_{11} := 1.0/\alpha$	
<hr/>	<hr/>
Continue with	Continue with
$\begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{pmatrix}$	$\begin{pmatrix} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{pmatrix}$
endwhile	endwhile

Fig. 1. Unblocked and blocked algorithms (left and right, respectively) for the inversion of a general matrix via GJE. Here, $m(\cdot)$ returns the number of rows of its argument.

3.1 Optimization

Look-ahead is a high performance technique to overcome performance bottlenecks due to the execution of serial phases during the computation of dense linear algebra operations [17]. This strategy is applied in tuned linear algebra libraries (Intel MKL, AMD ACML, the HPL Linpack benchmark, etc.) to overlap the factorization of a block panel in the LU and QR factorizations with the update of the remaining parts of the matrix. The resulting procedure yields superior performance at the cost of higher programming complexity, especially if several levels of look-ahead are simultaneously applied. Alternatively, a dynamic variant of look-ahead can be easily attained by employing a runtime like, e.g., SMPSSs [16,3] to schedule a dense linear algebra operation decomposed into a number of tasks with dependencies among them.

Consider the specific case of matrix inversion via the blocked GJE-based matrix inversion procedure in Figure 1 (right), and the partitioning

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right) = \left(\begin{array}{c|c|c|c} A_0 & A_1 & A_2^L & A_2^R \end{array} \right),$$

set at the beginning of the first iteration of the loop, where A_{TL}, A_{00} are both 0×0 , correspondingly A_0 contains no columns, and A_1 has b columns. Assume that A_2^L also contains b columns. The idea of look-ahead is, during this first iteration, to factorize the panel A_1 and, immediately after, perform the corresponding update of A_2^L ; next, the updates of A_0 and A_2^R are overlapped with the factorization of A_2^L during this initial iteration. One can then apply the same idea, during subsequent iterations $k = 2, 3, \dots$, to overlap the factorization of the $(k+1)$ -th block panel with the updates corresponding to the k -th iteration.

The appealing property of the GJE-based matrix inversion is that the amount of flops per iteration remains constant, easily accommodating look-ahead and enhancing its performance advantage compared with factorizations that operate on a decreasing number of data (e.g., LU, QR or Cholesky). This property can be also leveraged to obtain a balanced distribution of the computations between the CPU and the GPU in a heterogeneous platform. In particular, due to the complexity of the panel factorization (especially when partial pivoting is applied), it is more convenient to perform this operation on the CPU. On the other hand, the (left and right) updates can be off-loaded to the GPU except, possibly, for a small panel that can be computed on the CPU.

Consider now the data transfers required in this hybrid CPU-GPU matrix inversion procedure. We can initially transfer the full matrix A from the CPU (memory), via the PCI-e, to the GPU (memory). Given that this requires $n \times n$ memory operations (memops) for $2n^3$ flops, the communication cost is negligible for large n .

Consider next the data transfers that occur during the inversion procedure. At each iteration of the algorithm, a panel of b columns of A has to be transferred from GPU to CPU, factorized there, and the result has to be sent back to the GPU for the application of the corresponding updates. Thus, this requires

$2(n \times b)$ memops per iteration, which can be amortized with the $2n(n - b)b$ flops corresponding to the update provided b is chosen to be large enough (in practice, for optimal performance, $b \approx 700$ or larger).

Two basic optimization techniques can be applied to further enhance the performance of the hybrid CPU-GPU implementation. First, given that the block size b is moderately large, higher efficiency can be attained from the execution of the panel factorization on a multicore CPU if this is performed by using the blocked algorithmic procedure, with a block size $\hat{b} < b$ (and, usually, with values of 16 or 32 for \hat{b}). Second, the matrix-matrix products to be performed in the GPU as part of the update can be combined into operations of larger granularity, improving the FLOPS ratio in these architectures. In particular, consider for example the (left) update of the three blocks that compose A_0 . These operations can then be combined into (two matrix manipulation operations and) a single matrix-matrix product as follows:

$$\begin{aligned}\hat{A} &:= A_{10}, \quad A_{10} := 0 \\ \begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \end{pmatrix} &:= \begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \end{pmatrix} + \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} \hat{A}\end{aligned}$$

3.2 Experimental Evaluation

The routines for matrix inversion have been evaluated on a platform equipped with an Intel Xeon E5640 at 2.67GHz and a NVIDIA GeForce GTX 680 “Kepler”. Intel MKL (version 10.3.4), MAGMA (version 1.2.1) and NVIDIA CUBLAS (version 4.2.9) provided high-performance implementations of the necessary linear algebra kernels. We evaluate the performance of these routines in terms of GFLOPS (10^9 flops/sec.).

Figure 2 reports the performance obtained for the computation of the matrix inverse via three different routines and single-precision (SP) arithmetic. The black dashed line corresponds to the execution of LAPACK routines `sgetrf` + `sgetri` on the multicore processor. The red line shows the results obtained by the best GPU-based variant that computes the inverse using the LU factorization. This variant, named LU+GPU, employs a routine from the library MAGMA [13] to compute the LU factorization, an optimized routine developed by AICES-RWTH to obtain the inverse of the triangular matrix, and our *ad-hoc* implementation to solve the triangular system. (Previous results have demonstrated this as being the combination that delivers highest performance for an LU-based matrix inversion on GPUs [5].) The blue line shows the performance of the GJE-based routine on the GPU. The experimental results demonstrate the superior scalability of this last implementation. Although the LU-based implementation delivers a higher GFLOPS rate than the GJE-based counterpart for the inversion of matrices of dimension up to 5,000, for larger matrices, the GJE implementation offers the best results.

Figure 3 evaluates the same three routines using double-precision (DP) arithmetic in this case, showing that the GJE variant is consistently the best option

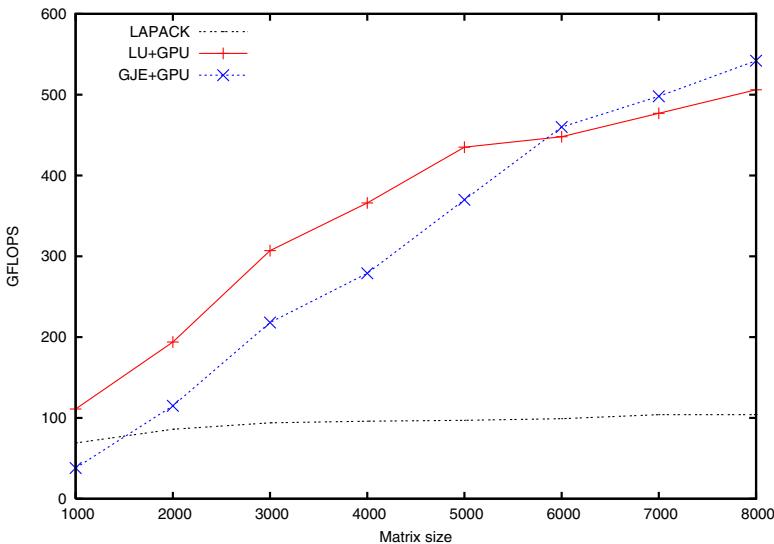


Fig. 2. Performance obtained for the inversion of matrices using SP arithmetic

regardless of the matrix dimension. The results in these experiments also illustrate the performance drop incurred by the introduction of DP, a factor between $2\times$ and $5\times$, motivating the next section.

4 MPIR for the Lyapunov Equation

Iterative refinement is a well-known technique to improve an approximate solution to a linear system of equations [11]. This technique has received renewed interest due to the performance gap between SP and DP in recent hardware accelerators, in particular, GPUs [4].

4.1 Refinement Procedure

Consider we have computed an initial, SP low-rank factor for the controllability Gramian of the Lyapunov $FW_c + W_cF^T + BB^T = 0$ via, e.g., the matrix sign function method. Hereafter, we will refer to this low-rank factor as L_0^S (so that $Y^S := L_0^S(L_0^S)^T$ is a SP approximation to the controllability Gramian W_c); and denote the corresponding computation/procedure as $L_0^S := \text{ApproxLyap}(F, B)$. (Note that, in principle, this initial factor could have been computed using some other numerical method different from the matrix sign function.) In [6], the following iterative procedure is introduced to refine this SP factor to the desired precision:

$$\mathcal{R}(L_k) := FL_kL_k^T + L_kL_k^TF^T + BB^T, \quad (5)$$

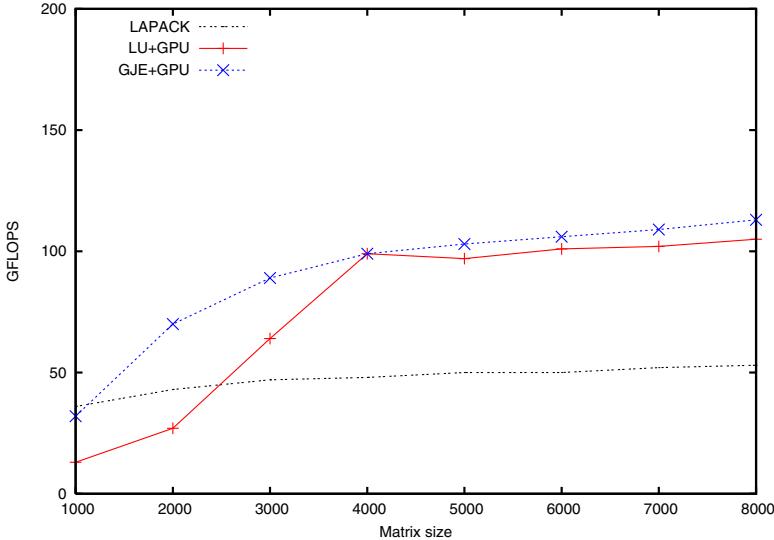


Fig. 3. Performance obtained for the inversion of matrices using DP arithmetic

$$\text{Decompose } \mathcal{R}(L_k) \rightarrow B_+ B_+^T - B_- B_-^T, \quad (6)$$

$$L_+ := \text{ApproxLyap}(F, B_+), \quad (7)$$

$$L_- := \text{ApproxLyap}(F, B_-), \quad (8)$$

$$Y_{k+1} := L_k L_k^T + L_+ L_+^T - L_- L_-^T, \quad (9)$$

$$\text{Decompose } Y_{k+1} \rightarrow L_{k+1}^T L_{k+1}^T - \hat{L}_- \hat{L}_-^T, \quad (10)$$

with $k = 0, 1, \dots$. In practice, (5) and (9) are never explicitly constructed. Also, (7) and (8) are computed using SP arithmetic; if the solution procedure is based on the matrix sign function, its computational cost becomes negligible in case the inverses that appear during the iteration are calculated once (e.g., during the initial solution $L_0^S := \text{ApproxLyap}(F, B)$), and saved for reuse during the refinement steps. Finally, (6) and (10) require DP arithmetic but are cheap to compute. For further details, see [6].

4.2 Experimental Evaluation

In this section we evaluate the MPIR approach in the platform described in section 3.2. For the experiments we employ the benchmark STEEL from the *Oberwolfach Model Reduction Benchmark Collection* [12]. This LTI system arises in a manufacturing method for steel profiles, where the objective is to design a control to assure the quality of the steel profile obtaining moderate temperature gradients while the rail is cooled down. In particular, the instance of the STEEL problem employed in this work has $n = 5,177$ state variables, $m = 7$ inputs and $p = 6$ outputs.

Table 1. Results obtained for the STEEL case using the MPIR technique

#Iter. Sign	#Iter. ref.	Residual	Time (s)
4	1	2.1e-10	6.7
4	2	6.5e-12	7.7
4	3	1.3e-13	8.9
4	4	2.9e-15	10.7

Table 1 presents the results obtained to solve the STEEL problem using the MPIR technique. The first and second columns show, respectively, the number of iterations of the sign function (in SP) and refinement iterations. Column 3 reports the residual $\|\mathcal{R}(L_{\bar{k}+k})\|_F / \|L_{\bar{k}+k} L_{\bar{k}+k}^T\|_F$, where $L_{\bar{k}+k}$ denotes the factor computed when k steps of iterative refinement are applied to the initial approximate factor computed after \bar{k} steps of the Newton iteration for the sign function; and column 4 corresponds to the execution time of the complete solver (sign function+MPIR) in seconds. The results demonstrate the moderate execution time added by MPIR. An accurate solution is obtained with 4 sign function iterations followed by 4 refinement steps, yielding an execution time of 10.7 seconds. A solution of similar accuracy using a DP implementation of the Newton iteration for the matrix sign function requires over 20 seconds.

5 Concluding Remarks

We have evaluated the use of two optimization techniques for control theory problems, namely, look-ahead and MPIR, on the new Kepler architecture. Look-ahead facilitates the concurrent execution of operations, allowing the concurrent use of many computational units, e.g., during matrix inversion via GJE on CPU-GPU platforms. The MPIR technique delivers DP accuracy while performing most of the computations in SP arithmetic. This approach is specially appealing on current GPUs, where SP performance is between 4 and 5× faster than DP. Experimental results show convenience of both techniques, demonstrating clear performance gains for the solution of control theory problems on heterogeneous platforms equipped with a GPU.

Acknowledgements. The researchers from the Universidad Jaume I (UJI) were supported by project TIN2011-23283 and FEDER.

References

1. Abels, J., Benner, P.: DAREX – a collection of benchmark examples for discrete-time algebraic Riccati equations (version 2.0). SLICOT Working Note 1999-15 (November 1999), <http://www.slicot.org>
2. Antoulas, A.: Approximation of Large-Scale Dynamical Systems. SIAM Publications, Philadelphia (2005)

3. Badia, R.M., Herrero, J.R., Labarta, J., Pérez, J.M., Quintana-Ortí, E.S., Quintana-Ortí, G.: Parallelizing dense and banded linear algebra libraries using SMPSS. *Concurrency and Computation: Practice and Experience* 21(18), 2438–2456 (2009)
4. Barrachina, S., Castillo, M., Igual, F.D., Mayo, R., Quintana-Ortí, E.S., Quintana-Ortí, G.: Exploiting the capabilities of modern GPUs for dense matrix computations. *Concurrency and Computation: Practice and Experience* 21, 2457–2477 (2009)
5. Benner, P., Ezzatti, P., Quintana-Ortí, E., Remón, A.: Matrix inversion on CPU-GPU platforms with application in control theory. *Concurrency and Computation: Practice and Experience* (to appear, 2012)
6. Benner, P., Ezzatti, P., Kressner, D., Quintana-Ortí, E., Remón, A.: A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU-GPU platforms. *Parallel Computing* 37(8), 439–450 (2011)
7. Benner, P., Quintana-Ortí, E., Quintana-Ortí, G.: State-space truncation methods for parallel model reduction of large-scale systems. *Parallel Computing* 29, 1701–1722 (2003)
8. Benner, P., Quintana-Ortí, E., Quintana-Ortí, G.: Solving linear-quadratic optimal control problems on parallel computers. *Optimization Methods Software* 23(6), 879–909 (2008)
9. Chahlaoui, Y., Van Dooren, P.: A collection of benchmark examples for model reduction of linear time invariant dynamical systems. SLICOT Working Note 2002–2 (February 2002), <http://www.slicot.org>
10. Choi, J., Dongarra, J.J., Pozo, R., Walker, D.W.: ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In: *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pp. 120–127. IEEE Computer Society Press (1992)
11. Higham, N.J.: *Accuracy and Stability of Numerical Algorithms*, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2002)
12. IMTEK, Oberwolfach model reduction benchmark collection,
<http://www.imtek.de/simulation/benchmark/>
13. MAGMA project home page, <http://icl.cs.utk.edu/magma/>
14. Mehrmann, V.: *The Autonomous Linear Quadratic Control Problem, Theory and Numerical Solution*. LNCIS, vol. 163. Springer, Heidelberg (1991)
15. Roberts, J.: Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *Internat. J. Control* 32, 677–687 (1980); Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department (1971)
16. SMP superscalar project home page,
http://www.bsc.es/plantillaG.php?cat_id=385
17. Strazdins, P.: A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization. Technical Report TR-CS-98-07, Department of Computer Science, The Australian National University, Canberra 0200 ACT, Australia (1998)

clOpenCL - Supporting Distributed Heterogeneous Computing in HPC Clusters^{*}

Albano Alves¹, José Rufino¹, António Pina², and Luís Paulo Santos²

¹ Polytechnic Institute of Bragança, Bragança, Portugal

{albano,rufino}@ipb.pt

² University of Minho, Braga, Portugal

{pina,psantos}@di.uminho.pt

Abstract. Clusters that combine heterogeneous compute device architectures, coupled with novel programming models, have created a true alternative to traditional (homogeneous) cluster computing, allowing to leverage the performance of parallel applications. In this paper we introduce *clOpenCL*, a platform that supports the simple deployment and efficient running of OpenCL-based parallel applications that may span several cluster nodes, expanding the original single-node OpenCL model. *clOpenCL* is deployed through user level services, thus allowing OpenCL applications from different users to share the same cluster nodes and their compute devices. Data exchanges between distributed *clOpenCL* components rely on Open-MX, a high-performance communication library. We also present extensive experimental data and key conditions that must be addressed when exploiting *clOpenCL* with real applications.

Keywords: Heterogeneous/Cluster/GPGPU Computing, OpenCL.

1 Introduction

Clusters of heterogeneous computing nodes provide an opportunity to significantly increase the performance of parallel and High-Performance Computing (HPC) applications, by combining traditional multi-core CPUs coupled with accelerator devices, interconnected by high throughput and low latency networking technologies. However, developing efficient applications to run in clusters that integrate GPUs and other accelerators often requires a great effort, demanding programmers to follow complex development methodologies in order to suit algorithms and applications to the new heterogeneous parallel environment.

Cluster nodes with GPUs are usually exploited by an hybrid approach: MPI is used to distribute the application across multiple CPUs, and OpenCL or CUDA are used to run specific routines (kernels) on GPU(s) at each node [1,2]; more

* This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-010067.

complex approaches have also been experimented [3], where OpenMP exploits CPU parallelism inside each MPI process and CUDA takes advantage of GPUs.

The former examples show that porting single-node-multi-GPU applications to a multi-node-multi-GPU execution environment is not straightforward. Thus, the benefits of a platform that would allow to efficiently run the same and unmodified program, whether in a single-node-multi-accelerator or a multi-node-multi-accelerator scenario are obvious.

Our initial goal was to contribute to the efficient implementation of a parallel path tracer, by improving the BVH operation [4], but we now target a more general one: to boost the performance of existing solutions by exploiting the hardware of an heterogeneous cluster. In this paper we introduce *clOpenCL* (*cluster OpenCL*), an integrated, transparent and efficient approach to take advantage of compute devices spread across an heterogeneous cluster.

The remaining of the paper is organized as follows: section 2 reviews some related work; section 3 describes the architecture of *clOpenCL*; section 4 presents and discusses a preliminary evaluation scenario based on matrix multiplication; finally, section 5 concludes and points out directions for future work.

2 Scaling Up OpenCL

2.1 The OpenCL Model

OpenCL is an open standard for programming different kinds of computing devices [5]. An OpenCL application comprises a *host* program and a set of *kernels* intended to run on compute *devices*; the OpenCL specification defines a language for kernel programming, and an API for transferring data between the host and devices (and to execute kernels on the later). Currently there are three major implementations of the OpenCL specification, supporting different compute devices: i) AMD APP SDK (for CPUs and AMD GPUs), ii) Nvidia's implementation (for NVIDIA GPUs only) and iii) Intel OpenCL SDK (for CPUs only).

But OpenCL poses a major problem: applications can only utilize the *local devices* present on a single machine. Thus, the number of OpenCL devices available to an application may be rather limited. For instance, the number of PCIe bus slots in a machine limits the number of usable GPUs. And although it would be possible to increase the number of slots through the use of bus extenders, the only way to achieve a scalable platform is to use a cluster of nodes, each one with its own (limited) set of computing devices.

Since in the original OpenCL model an application runs on a single node, a new/modified model is then required for OpenCL applications to be able to use several nodes. In the new model, the host application must be able to transfer/share data to/with *remote devices*.

2.2 Related Work

Running unmodified OpenCL applications on clusters with GPUs has been a common goal of several projects. Different approaches have been undertaken,

but the fact is that none combines the simplicity and the performance required for heterogeneous clusters shared by distinct users (usually operated in batch).

The Many GPUs Package (MGP) [6] can run extended OpenMP, C++ and unmodified OpenCL applications transparently on clusters with many GPU devices. It provides a simple API and the illusion of a single (virtual) host with many GPUs (single-system-image). The whole system uses the MOSIX VCL layer [7] to create the abstraction of a global OpenCL platform combining all GPUs present in a cluster (the CPU part of the application runs in a single node). Communications rely on TCP sockets and only binaries are distributed.

The Hybrid OpenCL [8] project integrates the network layer in the OpenCL runtime, translating in a "bridge program" (service) per cluster node. The system was developed for a particular device independent OpenCL implementation (FOXC OpenCL) which currently supports only x86 CPUs, thus preventing it to exploit high performance GPUs. Data exchanges are RPC based.

dOpenCL (*distributed* OpenCL) [9] has resemblances to *clOpenCL*: both support transparent multi-node-multi-accelerator OpenCL applications and combine a wrapper client library with remote services. However, dOpenCL is oriented to general distributed environments, uses a TCP/UDP based communication framework, and devices may not be concurrently shared. In turn, *clOpenCL* targets HPC clusters, uses Open-MX to maximize the utilization of commodity Gigabit Ethernet links, and devices are fully shareable. Both approaches work on top of any OpenCL platform and so are able to exploit many device types.

CUDA applications may also benefit from similar approaches. With rCUDA [11], applications may use CUDA-compatible GPUs installed in remote computers as if they were local. rCUDA follows the client-server model: clients use a wrapper library and a GPU network service listens for TCP requests.

In GVirtuS [10], a different approach is taken, in order to fill the gap between in-house hosted computing clusters (equipped with custom devices) and pay-for-use high performance virtual clusters (deployed via public or private computing clouds). GVirtuS allows a virtual machine to access CUDA powered GPGPUs in a transparent way, with an overhead slightly greater than a real machine setup.

However, in addition to the limited number of compute devices that CUDA can handle (NVIDIA only), none approach takes explicit advantage of the high performance interconnection technologies available in modern clusters.

3 Our Approach: *clOpenCL*

3.1 General Concept

The *clOpenCL* platform comprises a wrapper library and a set of user-level daemons. Every call to an OpenCL primitive is intercepted by the wrapper library which redirects its execution to a specific daemon at a cluster node or to the local OpenCL runtime. *clOpenCL* daemons are simple OpenCL programs that handle remote calls and interact with local devices. A typical *clOpenCl* application starts running at a particular cluster node and will create OpenCl contexts, command queues, buffers, programs and kernels across all cluster nodes.

For the exchange of data between the wrapper library and remote daemons, we adopted Open-MX, an open-source message passing stack over generic Ethernet [12], which provides low-level communication mechanisms at user-level space and allows to achieve low latency communication and low CPU overhead.

Figure 1 presents a) the software/hardware layers of the host component of an *clOpenCL* application, and b) the *clOpenCL* operation model upon which the host component interacts with multiple compute devices (local or remote).

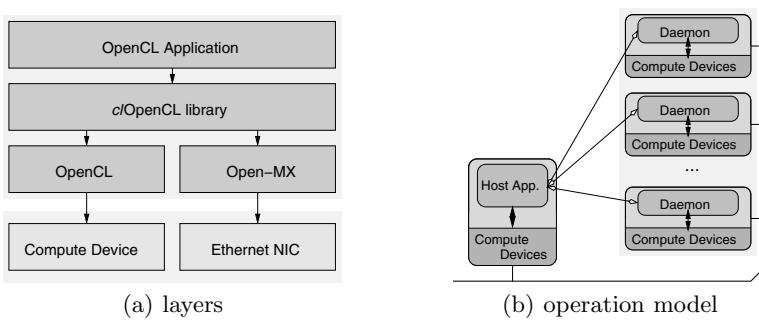


Fig. 1. *clOpenCL* architecture

3.2 Distributed Operation

Running a *clOpenCL* application requires the prior launching of *clOpenCL* per-user daemons at specific cluster nodes; each user may then choose the node (sub-)set to be effectively used by the application; different users may exploit distinct configurations, sharing or not particular nodes and compute devices.

When the host program starts, the *clOpenCL* wrapper library (which also wraps `main`) discovers all daemons by interacting with the Open-MX mapper service. This mapper creates a distributed directory that registers each process with an opened Open-MX end-point (along with the user id of the process owner).

In a traditional OpenCL application, the programmer has only to manipulate objects returned by the OpenCL API, namely: platform and device identifiers, contexts, command queues, buffers, images, programs, kernels, events and samplers. These objects are in fact pointers to complex OpenCL data structures, whose internals are hidden from the programmer. In a distributed/parallel environment, where OpenCL primitives are executed in multiple daemons, these pointers cannot be used to uniquely identify objects, because each daemon has its own address space. Thus, for each object created by OpenCL, the wrapper library returns a “fake pointer” used as a global identifier, and stores the real pointer along with the corresponding daemon location.

Each time the wrapper library redirects an OpenCL primitive, its parameters are packed in an Open-MX frame and sent to the remote daemon that will execute the primitive. Any parameters that reference OpenCL objects are previously mapped into their real pointers and the daemon is determined accordingly.

The library and daemons do not keep any information about calls to OpenCL primitives, *i.e.*, they are stateless. Any data needed for subsequent primitive calls is kept by the OpenCL runtime at each cluster node. As such there's no need to manage complex data structures related to OpenCL operation and state.

3.3 Using *clOpenCL*

Porting OpenCL programs to *clOpenCL* only requires linking with both the OpenCL and *clOpenCL* libraries. This is accomplished by taking advantage of the `-Xlinker --wrap` GCC directives for function wrapping in link-time.

Typically, an OpenCL application starts by querying the runtime for platforms and devices which, by design, are *local* (*i.e.*, provided by the node where the host application component is started). In *clOpenCL* such discovery returns the *local* platforms and devices (if any, once *clOpenCL* doesn't strictly require local ones), and also returns the set of *remote* platforms and devices provided by the cluster nodes where the user-specific *clOpenCL* services are running.

In order to know to which cluster node a certain platform belongs, we have extended the OpenCL primitive `clGetPlatformInfo` with the special attribute `CL_PLATFORM_HOSTNAME`. Having the possibility of selecting specific cluster nodes where to run the OpenCL kernels may be useful, *e.g.*, for load balancing.

Currently, we do not support the mapping of buffer and image objects. However, at its current state, the *clOpenCL* platform is enough to meet the purpose of testing its general concept, including running basic OpenCL applications.

4 Evaluation

4.1 Testbed Cluster

The testbed is a small Linux commodity cluster of 4 nodes, with an Intel Q9650 CPU (3GHz 4-core with 12 Mb of L2 cache), 8Gb of RAM (DDR3 1333MHz) and two Ethernet 1Gbps NICs (on-board Intel 82566DM-2 and a PCI64 SysKonnect SK-9871), per node. The nodes are also fitted with NVIDIA GTX 460 GPUs (1GB of GDDR5 RAM): one node (`node-0`) has 2 GPUs and three nodes (`node-{1,2,3}`) have 1 GPU each. The OpenCL platforms used were AMD SDK 2.6 (for CPU devices) and CUDA 4.1.28 (for the NVIDIA GPUs). OpenMX 1.5.2 was used with the SysKonnect NICs, interconnected via a dedicated ethernet network using a gigabit switch with jumbo frames (mtu 9000) enabled.

4.2 Test Application

We chose the *matrix product* as the test application because it's a simple and "embarrassingly parallel" case study, enough to verify the correctness and scalability of *clOpenCL* (our aim is not to offer a reference HPC implementation).

We narrowed our study to square matrices of order $n \in \{8K, 16K, 24K\}$ and single-precision (floats) elements. These 3 different orders were chosen to support

a minimal scalability study and, at the same time, to allow all the 3 matrices involved (the operands A and B , and the result $C = AB$) to be fully instantiated in the RAM (8Gb) of the node with the application host component (`node-0`); in the worst case ($n = 24K$) each matrix uses 2.25 Gb, for a total of 6.75 Gb.

The matrix product operation was parallelized using a block-based approach: A (B) was partitioned in horizontal (vertical) blocks $subA$ ($subB$) of order $slice \times n$ ($n \times slice$), so that $subC = subA subB$ is a block, of order $slice \times slice$, of C .

The OpenCL kernel used to produce a $subC$ block is shown in Figure 2. It is a “naive” implementation, not optimized to exploit advanced OpenCL features.

```
_kernel void sbmp(const int n, const int slice, __global float *subA,
                  __global float *subB, __global float *subC){
    int i, j, k; float v=0;

    i = get_global_id(0); j = get_global_id(1);
    for(k=0; k<n; k++)
        v += subA[i*n+k] * subB[j*n+k];
    subC[i*slice+j] = v;
}
```

Fig. 2. `sbmp` - a simple kernel for single-precision block-based matrix product

Producing a $subC$ requires $slice^2$ kernel executions (or work-items); this is achieved by the parameter `int global_work_size[2]={slice,slice}` of the OpenCL function that triggers the kernel execution (`clEnqueueNDRangeKernel`).

We set $slice = 1K, 2K, 4K$ when multiplying matrices of order $n = 8K, 16K, 24K$, respectively. The $slice$ values were chosen to allow any device to be able to fully store the triplet $<subA, subB, subC>$; in the worst case, with $n = 24K$ and $slice = 4K$, $subA$ and $subB$ need 384 Mb each and $subC$ needs 64 Mb, for a total of 832 Mb, still less than the 1 Gb of RAM of any GPU. At the same time, the values of $slice$ also ensure enough blocks for a fine-grain load-balancing among the various OpenCL devices; this comes from the observation that, in our cluster, a GPU is approximately twice as fast as a CPU when executing the `sbmp` kernel; thus, with 5 GPUs and 4 CPUS, we need at least $2 \times 5 + 4 = 14$ kernel runs (one per $subC$) to keep all devices busy; by requiring at least two kernel runs per device, we need at least 28 kernel runs overall; finally, $slice$ comes by solving $(n/slice)^2 \geq 28$ where $(n/slice)^2$ is the total number of kernel runs.

The test application is multi-threaded (PThreads) and follows a dynamic model of work (auto-)assignment: a thread is created for each OpenCL device involved in the matrix product; the threads select mutually exclusive pairs $<subA, subB>$, send them to their devices, trigger the kernel run and collect the results $subC$. Data transfers and kernel runs for remote devices are mediated by `clOpenCL` but, for local devices, they are direct (no daemon is involved).

4.3 Test Configurations

The test application was always started in the cluster node with the most performant set of OpenCL devices. That way, the utilization of that device set is maximized, once communication with its devices is purely local. In our testbed cluster, the node that fits this criterion is `node-0`, with 1 CPU and 2 GPUs.

Overall, under the constraints of our testbed cluster, there are 74 combinations of OpenCL devices, where `node-0` is always used (with at least one device), and zero or more remote nodes are used (with at least one device). All combinations were evaluated, but Table 1 shows only the subset that provides the best performance, for a certain number of CPUs (#C) and GPUs (#G) used; in each combination, the CPUs are denoted by C and the GPUs by G; they are represented in comma separated groups, with one group per each cluster node used to support the combination; in each combination, the 1st group of devices is from `node-0` and the next (eventual) groups are from `node-1` to `node-3` respectively.

Table 1. Performance Optimal Combinations of OpenCL Devices

#C \ #G	0	1	2	3	4	5
0		G	GG	GG,G	GG,G,G	GG,G,G,G
1	C	GC	GGC	GGC,G	GGC,G,G	GGC,G,G,G
2	C,C	GC,C	GGC,C	GGC,G,C	GGC,G,G,C	GGC,GC,G,G
3	C,C,C	GC,C,C	GGC,C,C	GGC,G,C,C	GGC,GC,G,C	GGC,GC,GC,G
4	C,C,C,C	GC,C,C,C	GGC,C,C,C	GGC,GC,C,C	GGC,GC,GC,C	GGC,GC,GC,GC

All combinations of Table 1 obey to the same general rule: they use the maximum possible number of local devices (on `node-0`) and scatter as much as possible the remote devices necessary (through `node-1` to `node-3`). Moreover, our tests showed that this rule is valid, regardless the order n of the matrices.

4.4 Test Results

Figure 3.a) represents the time took by the matrix product for $n=24K$, for all device combinations of Table 1. Times range from a maximum of ≈ 4800 s, when using a single local CPU (combination C), to a minimum of ≈ 469 s, when using all 5 GPUs of the cluster but only 3 CPUs (in the combination GGC,GC,GC,G)¹. Figure 3.b) zooms in Figure 3.a), for combinations where $\#C \geq 1$ and $\#G \geq 2$.

Figure 4.a) shows the speedups of all combinations, relative to combination GGC (with speedup = 1). Figure 4.b) zooms in figure 4.a) for $\#C \geq 1$ and $\#G \geq 2$.

The point of coordinates $\#C=1$ and $\#G=2$, that marks the lower end of the range defined by $\#C \geq 1$ and $\#G \geq 2$, translates to the combination GGC, meaning that all local devices of the starting node (`node-0`) are used. From such point onwards, the only way to increase performance is to use remote devices,

¹ Using all 4 CPUs (combination GGC,GC,GC,GC) takes ≈ 489 s.

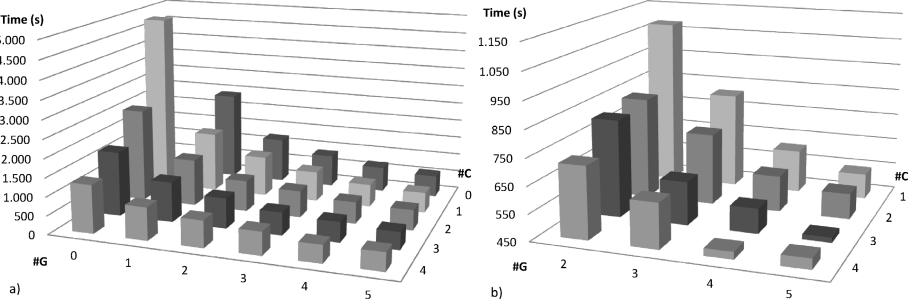


Fig. 3. Execution time for $n=24K$ (a) all combinations; b) $\#C \geq 1$ and $\#G \geq 2$)

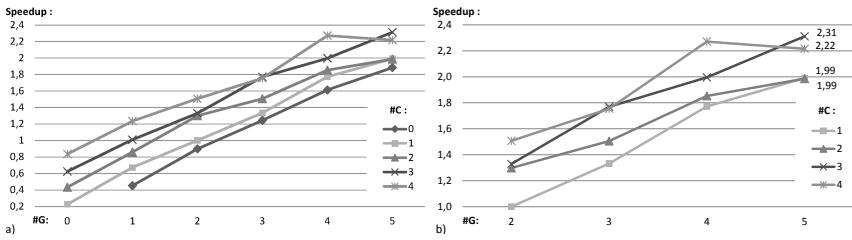


Fig. 4. Speedup for $n=24K$ (a) all combinations; b) $\#C \geq 1$ and $\#G \geq 2$)

but the speedups will be modest, as shown in Figure 4.b). Such demonstrates the importance of the combination GGC: if one wants to minimize the number of cluster nodes involved in the application execution, while maximizing performance, then using a node holding the device combination GGC is mandatory, and that node should be the one where the application starts. Moreover, GGC is the best scenario that a traditional OpenCL approach could exploit and so it makes sense to use it as the starting point to deploy *clOpenCL* applications.

Figures 5 to 6 show similar graphics for matrices of order $16K$ and $8K$. As expected, the matrix product times decrease (ranging from $\approx 1422s$ to $\approx 157s$ for $n = 16K$, and from $\approx 179s$ to $\approx 22s$ for $n = 8K$). Moreover, speedup values also decrease (with peak-values of 2.15 and 1.76 for orders $16K$ and $8K$, against 2.31 for order $24K$), showing that the scalability improves with the problem size. Differently from order $24K$, orders $16K$ and $8K$ achieve the best times when using all the 4 CPUs and 5 GPUs of the cluster (combination GGC,GC,GC,GC).

Real Speedup versus Ideal Speedup. An alternative (and perhaps more objective) measure of the merits of our approach results from the comparison of the maximum speedups achieved by *clOpenCl* (real speedups) with the maximum speedups theoretically achievable under ideal conditions (ideal speedups).

For order $24K$, we recall that the worst execution time is $T(C) \approx 4800s$, for a single CPU, and the best is $T(GGC,GC,GC,G) \approx 469s$, when 5 GPUs and 3 CPUs are involved. The evaluation of the execution times also reveals that a

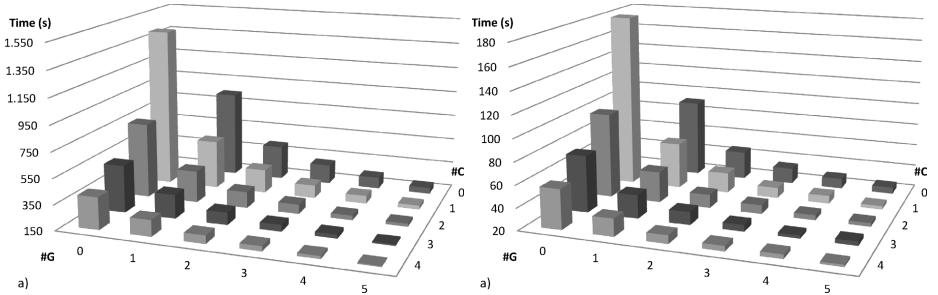


Fig. 5. Execution time (all combinations) for a) $n=16K$; b) $n=8K$

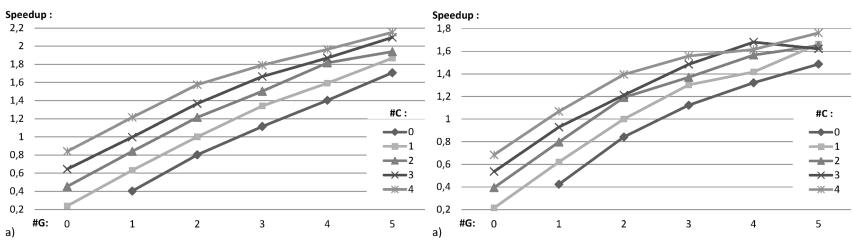


Fig. 6. Speedup (all combinations) for a) $n=16K$; b) $n=8K$

single GPU takes $T(G) \approx 2403s$, which is approximately half of $T(C)$. Thus, the time that a CPU takes to process a single work slice ($T_{slice}(C)$), is twice the time that a GPU would take ($T_{slice}(G)$). Or, conversely, the rate at which a GPU can process work slices ($\lambda_{slice}(G)$) is approximately twice the rate of a CPU ($\lambda_{slice}(C)$). So, if it were possible to host, in the same cluster node, 5 GPUs and 3 CPUs, then, for a time interval of duration $T_{slice}(C)$, we would have $5 \times 2 = 10$ work slices being processed by the 5 GPUs, and $3 \times 1 = 3$ work slices by the 3 CPUs, for a total of 13 work slices. It then follows that the maximum theoretical speedup is $S_{max}^{ideal} = 13$ (this assumes that there are enough work slices to keep all devices simultaneously busy). Now, the maximum effective speedup is $S_{max}^{real} = T(C) / T(GGC, GC, GC, G) = 10.23$. Finally, $S_{max}^{real} / S_{max}^{ideal} = 78.72\%$.

For $n=16K$, the worst and best execution times are $T(C) \approx 1422s$ and $T(GGC, GC, GC, GC) \approx 157s$, respectively. Also, $T(G) \approx 840s$, which is 59% of $T(C)$, relatively near from the 50% ratio achieved with $n=24K$. However, with 5 GPUs and 4 CPUs, the maximum theoretical speedup is now $S_{max}^{ideal} \approx 14$ and the maximum effective speedup is $S_{max}^{real} = T(C) / T(GGC, GC, GC, GC) = 9.05$. Then, $S_{max}^{real} / S_{max}^{ideal} = 64.69\%$, a smaller ratio in comparison to the one with $n=24K$.

Finally, for $n=8K$, the worst time is $T(C) \approx 179s$ and the best is $T(GGC, GC, GC, GC) \approx 22s$. Also, $T(G) \approx 91s$, which is $\approx 50\%$ of $T(C)$. Again, with 5 GPUs and 4 CPUs, the maximum theoretical speedup is $S_{max}^{ideal} \approx 14$. The maximum effective speedup is now $S_{max}^{real} = T(C) / T(GGC, GC, GC, GC) = 8.21$. Thus, $S_{max}^{real} / S_{max}^{ideal} = 58.68\%$, a smaller ratio in comparison to the one with $n=16K$.

The above $S_{max}^{real} / S_{max}^{ideal}$ ratios shed new light on the (somehow) modest speedups apparently achieved by *clOpenCL* against the best pure OpenCL scenario (GGC). In particular, they prove that for bigger problem sizes (as illustrated by the matrix product of order $24K$), *clOpenCL* exhibits good scalability.

5 Conclusions

We have presented the design and implementation details of a new platform that facilitates the execution of OpenCL applications in heterogeneous clusters. Other projects have also focused on the same objective, but *clOpenCL* has two main advantages: it is able to take full advantage of commodity networking hardware through Open-MX, and programmers/users do not need special privileges neither exclusive access to scarce resources to deploy the desired running environment.

We used an embarrassingly parallel program to evaluate *clOpenCL*, with different problem sizes and cluster device combinations. Results show that *clOpenCL* is useful for exploiting multi-node-multi-GPU environments: porting previous OpenCL applications is straightforward and performance gains are attractive.

In the future, we will expand the set of OpenCL primitives supported by *clOpenCL*. Also, adding BSD sockets as an alternative communication layer to Open-MX will allow for *clOpenCL* to be used in more distributed scenarios.

References

1. Lawlor, O.: Message Passing for GPGPU Clusters: cudaMPI. In: IEEE Cluster PPAC Workshop, pp. 1–8 (2009)
2. Stefanski, T., Chavannes, N., Kuster, N.: Hybrid OpenCL-MPI parallelization of the FDTD method. In: International Conference on Electromagnetics in Advanced Applications (ICEAA), pp. 1201–1204 (2011)
3. Yang, C.-T., Huang, C.-L., Lin, C.-F.: Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters. Computer Physics Communications 182, 266–269 (2011)
4. Goldsmith, J., Salmon, J.: Automatic creation of object hierarchies for ray tracing. IEEE Computer Graphics & Applications 7(5), 14–20 (1987)
5. Munshi, A.: The OpenCL Specification. Khronos OpenCL Working Group (2009)
6. Barak, A., Ben-nun, T., Levy, E., Shiloh, A.: A Package for OpenCL Based Heterogeneous Computing on Clusters with Many GPU Devices. Science, 1–7 (2010)
7. Barak, A., Shiloh, A.: The Virtual OpenCL (VCL) Cluster Platform. In: Proc. Intel European Research & Innovation Conference, p. 196 (2011)
8. Aoki, R., Oikawa, S., Nakamura, T., Miki, S.: Hybrid OpenCL: Enhancing OpenCL for Distributed Processing. In: IEEE 9th International Symposium on Parallel and Distributed Processing with Applications Workshops, pp. 149–154 (2011)
9. Kegel, P., Steuwer, M., Gorlatch, S.: dOpenCL: Towards a Uniform Programming Approach for Distributed Heterogeneous Multi-/Many-Core Systems. In: 26th IEEE Int. Parallel and Distributed Processing Symposium Workshops, pp. 174–186 (2012)

10. Giunta, F., Montella, R., Laccetti, G., Isaila, F., Blas, F.: A GPU Accelerated High Performance Cloud Computing Infrastructure for Grid Computing Based Virtual Environmental Laboratory. In: Advances in Grid Computing (2011)
11. Duato, J., Peña, A., Silla, F., Mayo, R., Quintana-Ortí, E.: Reducing the number of GPU-based accelerators in high performance clusters. In: International Conference on High Performance Computing and Simulation, pp. 224–231 (2010)
12. Goglin, B.: High-Performance Message Passing over generic Ethernet Hardware with Open-MX. Elsevier Journal of Parallel Comp. (PARCO) 37(2), 85–100 (2011)

Mastering Software Variant Explosion for GPU Accelerators

Richard Membarth¹, Frank Hannig¹, Jürgen Teich¹,
Mario Körner², and Wieland Eckert²

¹ Hardware/Software Co-Design, Department of Computer Science,
University of Erlangen-Nuremberg, Germany

{richard.membarth,hannig,teich}@cs.fau.de

² Siemens Healthcare Sector, H IM AX,
Forchheim, Germany

{mario.koerner,wieland.eckert}@siemens.com

Abstract. Mapping algorithms in an efficient way to the target hardware poses a challenge for algorithm designers. This is particular true for heterogeneous systems hosting accelerators like graphics cards. While algorithm developers have profound knowledge of the application domain, they often lack detailed insight into the underlying hardware of accelerators in order to exploit the provided processing power. Therefore, this paper introduces a rule-based, domain-specific optimization engine for generating the most appropriate code variant for different Graphics Processing Unit (GPU) accelerators. The optimization engine relies on knowledge fused from the application domain and the target architecture. The optimization engine is embedded into a framework that allows to design imaging algorithms in a Domain-Specific Language (DSL). We show that this allows to have one common description of an algorithm in the DSL and select the optimal target code variant for different GPU accelerators and target languages like CUDA and OpenCL.

1 Introduction

Computer systems are increasingly heterogeneous, as many important computational tasks, such as multimedia processing, can be *accelerated* by special-purpose processors that outperform general-purpose processors by one or two orders of magnitude, importantly, in terms of energy efficiency as well as in terms of execution speed.

Until recently, every accelerator vendor provided its own Application Programming Interface (API), typically based on the C language. For example, NVIDIA's API, called CUDA, targets systems accelerated with GPUs. Ryoo et al. [9] highlight the complexity of CUDA programming, in particular, the need for exploring thoroughly the space of possible implementations and configuration options. OpenCL, a new industry-backed standard API that inherits many traits from CUDA, aims to provide software portability across heterogeneous systems: correct OpenCL programs will run on any standard-compliant implementation. OpenCL per se, however, does not address the problem of *performance portability*; that is, OpenCL code optimized for one accelerator device

may perform dismally on another, since performance may significantly depend on low-level details, such as data layout and iteration space mapping [3].

To solve this problem, we focus on code generation and customization for GPU accelerators from the same code base in this paper. The support of heterogeneous systems with GPU accelerators from possibly different vendors is crucial. There are many ways to customize a program. Commonly, a program uses command-line parameters, conditional compilation, or aspect-oriented programming [2]. An alternative is product-line engineering, where tailored software variants are generated based on the specification as stated by the user or programmer [8].

This paper introduces a rule-based, domain-specific and target-specific optimization engine that generates the most appropriate software variant for a given algorithm description in a DSL for the desired target GPU accelerator. While we have previously shown that we can generate target code for CUDA and OpenCL using the Heterogeneous Image Processing Acceleration (HIPA^{cc}) framework that leverages different architectural features (such as memory alignment, locality, unrolling, or tiling) [6, 7], there is still the question *how to select the most appropriate code variant for a given input algorithm on a particular target platform?* The proposed optimization engine builds on software product-line technology, separating the user-visible *features* (described in the DSL) from the implementation *artifacts* (generated by a source-to-source compiler).

The remainder of the paper is organized as follows: Section 2 introduces the features supported by the HIPA^{cc} framework to design image processing algorithms. The different software variants due to variety in mapping features to different target hardware as well as due to diversity in optimization possibilities are described in Section 3. The following Section 4 describes the domain-specific and target-specific artifact selection provided by HIPA^{cc} and evaluates the proposed rule-based optimization engine. Finally, the paper is concluded in Section 5.

2 Domain-Specific Features in Medical Image Processing

In this section, domain-specific *features* from the considered domain of medical imaging are presented, showing the variability of applications designed by the programmer. Thereby, a *feature* specifies a end-user visible characteristic of a system [8]. Based on the *features* selected by the programmer, different variants of image processing kernels can be generated.

We have identified and implemented domain-specific *features* in the HIPA^{cc} framework in a DSL. From this description, the HIPA^{cc} framework generates low-level, optimized CUDA and OpenCL target code for execution on GPU accelerators as depicted in Fig. 1. It was shown that efficient code can be generated for *features* captured in the DSL [6, 7]. In addition to the CUDA and OpenCL back ends for GPU accelerators, also back ends for host systems are provided—mainly for debugging or as fall-back (that is, no thorough optimizations are applied within these back ends).

The HIPA^{cc} framework supports *features* that describe the desired behavior of the application. This includes a) an *Image*, representing the data storage for image pixels (e. g., represented as integer number or floating point number), b) an *Iteration Space*, defining a rectangular region of interest in the output *Image*. Each pixel within this

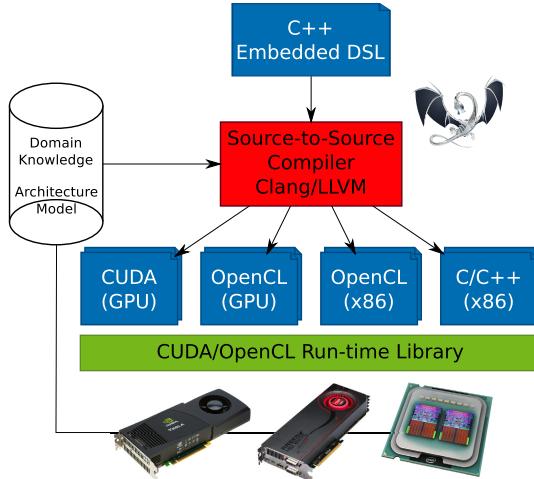


Fig. 1. HIPA^{cc} framework: efficient low-level code generation based on a DSL. Optimizations and transformations are applied according to the target hardware and domain knowledge.

region is generated by c) an *Operator*. There exist three types of operators, *Point Operators*, *Local Operators*, and *Global Operators*. To read from an *Image*, d) an *Accessor* is required, defining how and which pixels are *seen* within the *Operator*. Similar to an *Iteration Space*, the *Accessor* defines the pixels that are read by the *Operator*. At least these *features* are required to design an image processing kernel. In addition, further features are available, for example, a *Mask* can store the filter mask constants for a *Local Operator* or a *Convolution*—a special instance of the *Local Operator*. Since *Local Operators* read neighboring pixels, out-of-bounds memory accesses occur and a *Boundary Condition* is required to define how to handle them: this includes to return a *constant* value, to update the index such that either the index is *clamped* at the border or the image is virtually *repeated* or *mirrored* at the border. Also not to care for out-of-bounds memory accesses is a valid option—resulting in an *undefined* behavior. In case images of different resolution are used within an *Operator*, each output pixel in the *Iteration Space* is mapped to a corresponding pixel in each input image. Consider downscaling an image by a factor of two in each dimension: only one out of four pixels would contribute to the output image when the *Iteration Space* is mapped (scaled) to the input image. Therefore, *Interpolation* can be defined for an *Accessor*, such that the neighboring pixels of the mapped iteration point define the pixel according to an algorithm. Supported are interpolation using *nearest neighbor*, *linear filtering*, *bicubic* filtering, and *Lanczos* filtering.

Fig. 2 shows all domain-specific *features* supported by the HIPA^{cc} framework. The programmer can select any combination of these *features* to define all possible kinds of image processing algorithms: varying number of input images, different operators, processing of only a part of the input or output image, defining boundary conditions or interpolations, etc.

3 Domain-Specific and Target-Specific Variant Selection

While the previous section introduced the domain-specific *features* that can be used by the programmer, the implementation possibilities are shown here. Each implementation is a (reusable) *artifact* of a *feature* and defines how a *feature* is realized in the target language [8]. That is, each optimization that is applied when a *feature* gets implemented, results in a new software variant.

In the proposed rule-based optimization engine *artifacts* are automatically generated by a source-to-source compiler, transforming and translating the intermediate representation of the algorithm described in the DSL, before target code is emitted by one of the back ends. The optimization engine differentiates between a) domain-specific *artifacts* and b) target-specific *artifacts*. While the first category implements a *feature*, the second class realizes a *feature* in an efficient way by providing a target-specific implementation. Fig. 3 shows all *artifacts* supported by the HIPAcc framework.

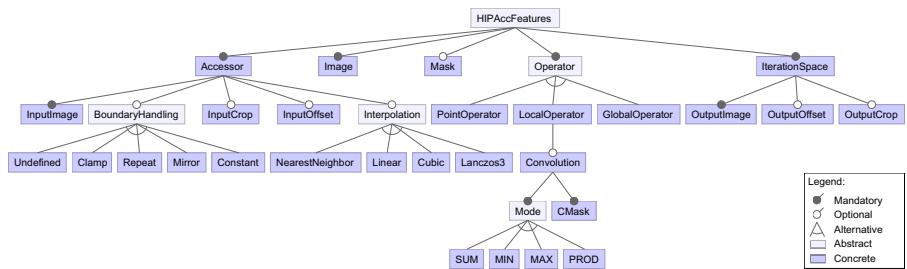


Fig. 2. Supported domain-specific *features* for image processing

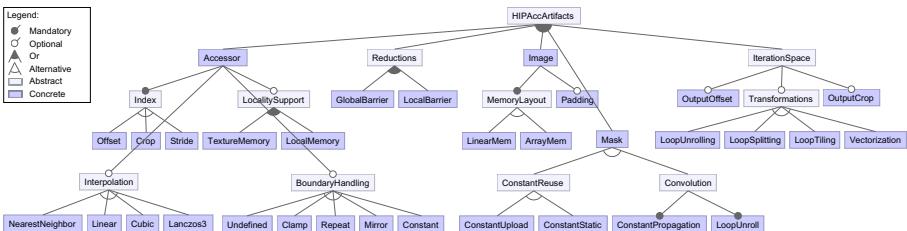


Fig. 3. Supported artifacts and optimizations for image processing

3.1 Domain-Specific Artifacts

The domain-specific artifacts implement features that are implied by their specification. These include the offset and crop of an *Accessor* or *Iteration Space*, which are considered when the index is calculated to access pixels. Similar, a *Boundary Condition* is realized by adding conditionals that check if the image is accessed out-of-bounds and *Interpolation* by returning the interpolated pixel value after mapping the iteration point to the region of interest.

Table 1. Domain-specific artifacts/optimizations for image processing kernels

DSL-Feature	Artifact/Optimization
<i>Image</i>	Linear memory layout
<i>Mask</i>	Upload constants
<i>Convolution Mask</i>	Propagate constants & unroll loops
<i>Boundary Condition</i>	Split loops/iteration space

In addition, target-independent *artifacts* are selected that optimize the performance of the generated target code, but do not change the characteristic of the application. Table 1 lists these *features* and the selected *artifacts* by our optimization engine. These are: for an *Image*, linear memory layout is preferred over array memory due to less overhead. The constants of a filter mask are uploaded at run-time and not stored to a statically initialized array on the device if used within a local operator. If the filter mask is part of a convolution, the convolution is unrolled and the filter mask constants are propagated. For boundary handling, different code variants of the kernel are generated (loop splitting). Each variant includes only conditionals for the top-left, top, top-right, etc. image border. For the vast majority of pixels a code variant without boundary handling is used. At run-time the appropriate code variant is selected by each compute unit on the GPU.

For instance, *uCLbench* [10] shows that kernels using statically allocated local memory are faster than the same kernels using dynamically allocated local memory. We encountered the opposite behavior when reading data from constant memory: in case the constant memory is stored to statically allocated device memory, execution takes longer compared to uploading the constants at run-time. What happens here is that in the former case the run-time system uploads the constants at execution time before the kernel is started. In the latter case, the constants can be uploaded during execution of another kernel or during initialization. That way, the upload can be hidden and this artifact is consequently selected by the proposed optimization engine.

3.2 Target-Specific Artifacts

The second category is more interesting and allows to generate applications tailored for heterogeneous architectures: target-specific code is generated based on domain knowledge and an architecture model. Here, the *artifacts* are selected according to a rule-based, target-specific optimization engine that selects the most appropriate *artifact* for a given target platform due to superior performance of the code variant or due to the use of instructions supported only by some target platforms. The target-specific artifacts are as follows:

- *Image*: a) the amount of padding, or the best alignment for the first pixel in each line; b) the use of texture memory or local memory such that for example pixels loaded by thread i can be reused by neighboring threads $i - 1$ and $i + 1$. The decision for this locality support depends in turn on a feature (i. e., the presence of a local operator).

- *Global Operator*: global barriers are not supported by all target devices.
- *Iteration Space*: a) the loop unrolling factor (changing the iteration space), that is, how many output pixels are calculated by one thread. Again, this factor depends also on other features (i. e., which operator do we generate code for?); b) the tiling of the iteration space: how many pixels are assigned to one compute unit; and c) whether vectorization¹ should be enabled for code generation (which changes also the iteration space).

4 Evaluation and Discussion

In this section, the selection of target-specific artifacts and optimizations is evaluated. The decision what optimization should be applied for what target (hardware and back end) is based on a) our previous results [6, 7], b) micro-benchmarks available online for CUDA [11] and OpenCL [10], and c) our own micro-benchmarks for typical kernel candidates from the medical domain. Our previous results show that a multitude of different software variants (with varying performance) can be generated using the HIPA^{cc} framework. Good variants have a competitive performance (OpenCV, RapidMind, and manual implementations) and are used to create the rules for the proposed optimization engine.

4.1 Evaluation

All (micro-)benchmarks have been executed on four GPU architectures, representing the two most recent GPU architectures from AMD and NVIDIA. Fig. 4 shows exemplarily the influence of exploiting locality (texture memory and local memory) for the Gaussian image filter (implemented as a local operator) for different filter window sizes. The results give several interesting insights: using texture memory yields the best results for CUDA on both cards from NVIDIA, once in combination with local memory (Fermi) and once without local memory support (Tesla). However, using texture memory for OpenCL is slower compared to reading directly from global memory. This can be attributed to the fact that CUDA supports linear texture memory while OpenCL provides only texture arrays. Linear texture memory is faster for the access pattern of local operators and was, hence, used in the CUDA back end. It is surprising that using texture memory in combination with local memory is still faster on the Fermi card (Tesla C2050), since Fermi introduced L2 caches that should make textures obsolete. On the cards from AMD (OpenCL) we see also the influence of the filter window size: for small filter sizes, texture memory and global memory provide the best performance. Only for big filter sizes (> 17 on the Radeon HD 5870 and > 21 on the Radeon HD 6970), the benefit of local memory is noticeable and results in faster execution. Here, the optimal implementation can be selected depending on the problem size (i. e., the filter window size).

Table 2 summarizes the target-specific artifacts and optimizations that are part of our rule-based optimization engine. Just to emphasize two artifacts: first, the unroll

¹ Vectorization is not yet supported by the framework.

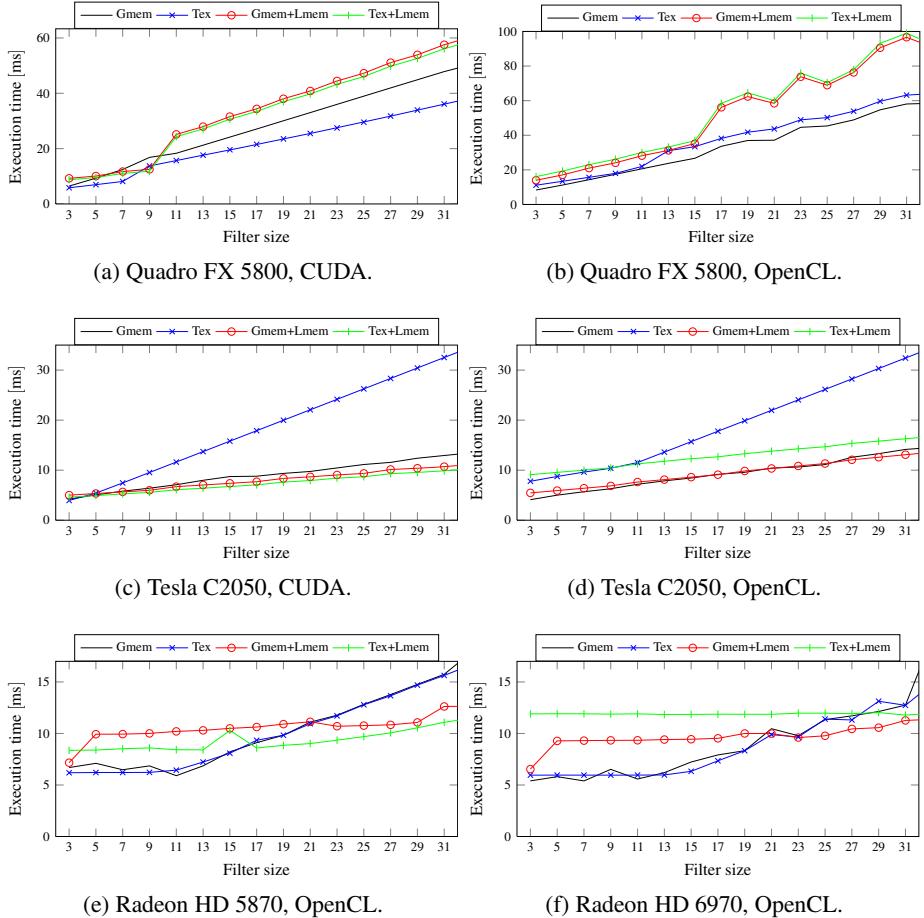


Fig. 4. Execution time of Gaussian image filter for different filter sizes, using different code variants to exploit locality (texture memory and local memory) for the CUDA and OpenCL back ends in ms for an image of 4096×4096 pixels. The configuration for all cards was 32×4 threads, only for the Tesla C2050 a configuration of 32×8 threads was used. No unrolling was applied.

factor for the iteration space improves performance across all GPUs, but with different parameters—depending on the kernel characteristics (point, local, or global operator). For example, on the Tesla C2050, performance drops on local operators when loop unrolling is applied, but improves by more than 20 % when applied for local operators. For global operators, the performance improves significantly when unrolling by a factor of 15 or 17, but drops again by 10 % for an unroll factor of 16. Second, target-specific instructions like the global barrier `__threadfence()`, which is only supported when generating CUDA code on the Tesla C2050, are utilized.

Table 2. Target-specific artifacts/optimizations for the Evergreen, Northern Islands, Tesla, and Fermi GPU architectures for the CUDA/OpenCL back ends

	Evergreen	Northern Islands	Tesla	Fermi
Radeon HD 5870	Radeon HD 6970	Quadro FX 5800	Tesla C2050	
Alignment (bytes)	1024	256	256	256
Vectorization [†]	on	on	off	off
Tiling [‡]	heuristic	heuristic	heuristic	heuristic
Point Operators				
Unroll factor	4	4	8/8	1/1
Texture memory	off	off	off	off
Local memory	off	off	off	off
Local Operators				
Unroll factor	1	1	16/2	8/32
Texture memory	on	on	on/off	on/off
Local memory	off	off	off	on
Global Operators				
Unroll factor	32	32	31/31	15/15
Texture memory	off	off	off	off
Local memory	on	on	on	on
Local barrier	on	on	on	on
Global barrier	off	off	off	on/off

[†] Vectorization is not yet implemented, however, it can be anticipated that AMD's VLIW4/5 architectures benefit from vectorization.

[‡] See [7] for details.

4.2 Discussion

To generate rules for the expert system—to decide which code variant to generate—variant exploration is supported by the HIPA^{cc} framework. Invoking the source-to-source compiler, the internal expert system decides what code variant to use. However, the compiler allows also to set artifacts manually using compiler switches. For example, the user can specify that local memory or texture memory should be turned on or off. Similar, the amount of padding or the unroll factor can be set by the user. This allows easy exploration of features for future architectures. In addition, the source-to-source compiler can generate code for exploring tiling of the generated kernels using the `--explore-config` compiler switch. The generated code includes macro-based code snippets for features that depend on tiling (statically allocated local memory and the block size). When executed, the run-time system compiles those variants using just-in-time compilation for both CUDA and OpenCL. Together with the `--time-kernels` compiler flag, which takes ten runs into account to determine the execution time, this allows to generate the required expert knowledge for new architectures and to validate the used heuristics.

Looking at the supported artifacts and optimizations in Fig. 3, it is clear that the created *variant-space* that can be generated explodes and it is often even not clear what artifact yields the best performance. Consider only the different possibilities for storing and reading pixels, which are not specified by any feature and can be chosen by the optimization engine: any combination of a) linear memory layout or array memory layout, b) padding, and c) global memory or texture memory, and d) local memory can be selected. This results in 16 variants (assuming that the amount of padding is known) just for how data is stored and accessed—not considering the combinations when offset, crop, interpolation, or boundary handling is required. This are the variants for one single image, but this can be different for each image, resulting in $n \cdot 16$ variants for n images. For optimizations with a parameter such as unrolling or padding, each valid value for the parameter (e. g., unroll factor $\in 2, \dots, height$) results in another variant. Even worse, different GPU accelerators require different optimizations, adding another dimension to the optimization space. Exploring these variants by hand is not feasible. Therefore, the proposed rule-based optimization engine helps to manage the explosion of code variants for GPU accelerators.

4.3 Related Work

Tuning code to a particular architecture is found in many software frameworks. Compilers create optimized code for a target Instruction Set Architecture (ISA) and apply transformations tuned for the target architecture. More specialized approaches use model-based optimizations, auto-tuning, or machine learning to select the most appropriate code variant for a particular platform. In machine learning [4], the run-time system explores different software variants at run-time (e. g., when the system is idle) in order to use that information for future execution on the platform. Since our target systems perform time-critical applications, machine learning is no option. In auto-tuning, different variants are explored at installation-time to determine the best tile size, loop unroll factor etc. Therefore, templates are instantiated and evaluated for a particular problem like Basic Linear Algebra Subprograms (BLAS). An examples for this approach is ATLAS [1] where the compute intensive algorithms are tuned for the target platform. Similar, it was shown that General Matrix Multiply (GEMM) can be tuned efficiently for GPU accelerators using auto-tuning based on templates [5]. However, Yotov et al. showed that using model-driven optimizations, near-optimal values for the optimization parameters can be selected within ATLAS [12].

In this paper, we focus similar to Yotov et al. on a rule-based optimization engine to generate target-specific code variants that can be used for heterogeneous systems. The optimization engine decides based on an expert system what code variant to generate. Moreover, the expert system used in our approach is built upon domain-knowledge and an architecture model for the target architecture, fusing application-, domain-, and architecture-knowledge. The rules for the expert system are based on micro-benchmarks for different typical kernel candidates in the considered domain.

5 Conclusions

In this paper, we presented a rule-based, domain-specific optimization engine, for generating the most appropriate code variant for different GPU accelerators.

The optimization engine does not only rely on knowledge of the target accelerator in order to decide what code variant to use. Instead, it fuses domain- and architecture knowledge, which allows to select the optimal code variant not only with respect to the target accelerator, but also to the algorithm description in the DSL. All transformations and optimizations are transparent to the programmer and are performed on the basis of the same DSL description. It was shown that using the proposed optimization engine, CUDA and OpenCL software variants can be generated with competitive performance compared to the GPU back end of the widely used image processing library OpenCV, RapidMind, or even hand-tuned implementations.

The presented domain-specific optimization engine has been implemented in the HIPA^{cc} framework, which is available as open-source under <https://sourceforge.net/projects/hipacc>.

References

1. Clint Whaley, R., Petitet, A., Dongarra, J.: Automated Empirical Optimizations of Software and the ATLAS Project. *Parallel Computing* 27(1), 3–35 (2001)
2. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)
3. Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., Dongarra, J.: From CUDA to OpenCL: Towards a Performance-portable Solution for Multi-platform GPU Programming. *Parallel Computing* 38(8), 391–407 (2011)
4. Grewe, D., Wang, Z., O’Boyle, M.F.: A Workload-Aware Mapping Approach for Data-Parallel Programs. In: Proceedings of the 6th International Conference on High-Performance and Embedded Architectures and Compilers, HiPEAC, pp. 117–126. ACM (January 2011)
5. Li, Y., Dongarra, J., Tomov, S.: A Note on Auto-tuning GEMM for GPUs. In: Allen, G., Nabrzyski, J., Seidel, E., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2009, Part I. LNCS, vol. 5544, pp. 884–892. Springer, Heidelberg (2009)
6. Membarth, R., Hannig, F., Teich, J., Körner, M., Eckert, W.: Automatic Optimization of In-Flight Memory Transactions for GPU Accelerators based on a Domain-Specific Language for Medical Imaging. In: Proceedings of the 11th International Symposium on Parallel and Distributed Computing, ISPDC. IEEE (June 2012)
7. Membarth, R., Hannig, F., Teich, J., Körner, M., Eckert, W.: Generating Device-specific GPU Code for Local Operators in Medical Imaging. In: Proceedings of the 26th IEEE International Parallel & Distributed Processing Symposium, IPDPS, pp. 569–581. IEEE (May 2012)
8. Pohl, K., Böckle, G., Van Der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer (2005)
9. Ryoo, S., Rodrigues, C., Stone, S., Stratton, J., Ueng, S., Baghsorkhi, S., Hwu, W.: Program Optimization Carving for GPU Computing. *Journal of Parallel and Distributed Computing* 68(10), 1389–1401 (2008)
10. Thoman, P., Kofler, K., Studt, H., Thomson, J., Fahringer, T.: Automatic OpenCL Device Characterization: Guiding Optimized Kernel Design. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011, Part II. LNCS, vol. 6853, pp. 438–452. Springer, Heidelberg (2011)
11. Wong, H., Papadopoulou, M., Sadooghi-Alvandi, M., Moshovos, A.: Demystifying GPU Microarchitecture through Microbenchmarking. In: Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, pp. 235–246. IEEE (2010)
12. Yotov, K., Li, X., Ren, G., Garzaran, M., Padua, D., Pingali, K., Stodghill, P.: Is Search Really Necessary to Generate High-performance BLAS? *Proceedings of the IEEE Special Issue on “Program Generation, Optimization, and Platform Adaptation”* 93(2), 358–386 (2005)

Exploring Heterogeneous Scheduling Using the Task-Centric Programming Model

Artur Podobas, Mats Brorsson, and Vladimir Vlassov

Royal Institute of Technology, KTH
`{podobas, matsbror, vladv}@kth.se`

Abstract. Computer architecture technology is moving towards more heterogeneous solutions, which will contain a number of processing units with different capabilities that may increase the performance of the system as a whole. However, with increased performance comes increased complexity; complexity that is now barely handled in homogeneous multiprocessing systems. The present study tries to solve a small piece of the heterogeneous puzzle; how can we exploit all system resources in a performance-effective and user-friendly way? Our proposed solution includes a run-time system capable of using a variety of different heterogeneous components while providing the user with the already familiar task-centric programming model interface. Furthermore, when dealing with non-uniform workloads, we show that traditional approaches based on centralized or work-stealing queue algorithms do not work well and propose a scheduling algorithm based on trend analysis to distribute work in a performance-effective way across resources.

Keywords: Task-Centric run-time systems, Heterogeneous computing, TilePRO64 tasks, GPU tasks, OmpSs.

1 Introduction

As technology advances computers are becoming more and more difficult to program; especially when aiming to utilize all the chips' features. The current direction concerning processor architecture is to accumulate as many processor cores as possibly on the chip to get so-called multi- or many-core processors. Primarily this direction is driven by the power and thermal limitations of the technology scaling; more transistors on a smaller area create larger power densities putting greater stress on the cooling mechanisms. Furthermore, multi- and many-core processors are slowly evolving to be more heterogeneous in nature. And hardware vendors are already starting to tape-out heterogeneous devices such as the AMD Fusion and ARM bigLITTLE. We cannot say much about the future but it is likely that the current trend of increasing heterogeneity will continue; and software developers are already now struggling keeping the pace with homogeneous multi- and many-core technology. How do we write parallel software that targets several heterogeneous systems so that it is portable, scalable and user-friendly? One solution is to provide the software programmer with a set of libraries that maintains and handles the parallelism that the programmer exposes. The library should provide functions for exposing and synchronizing parallel work. A programming paradigm that

has the potential of supporting these features is called the *task-centric* programming paradigm. The task-centric paradigm abstracts the user away from managing threads. Instead, the programmer focuses on exposing parallel work in the form of tasks. Tasks can be of any granularity, ranging from an entire application to a simple calculation. To clarify this further, a task is a sequential part of the application that can be executed concurrently with other tasks. Tasks are dynamic in nature; a task can create several new tasks to further decompose the work and increase the parallelism. There are many benefits of choosing a task-centric over a thread-centric framework, such as improved load-balancing (resource-utilization), portability and user-friendliness. Notable task-centric libraries include: Cilk+ (based on Cilk-5 [7]) focusing on determinism, Nanos++ on user friendliness and versatility, Threading-Build Block on Object-Oriented-Programming, Wool [6] on fine-grained parallelism, StarPU [1] on GPUs. In the present paper, our primary contributions are:

- A task-centric run-time system capable of handling several types of heterogeneous processing units with distributed memory spaces, clock frequency, core count and ISA.
- A scheduling algorithm that is based around linear regression with user-provided feedback concerning tasks' properties. The regression technique is very dynamic; it is continuously calculated and used on-line while the application is running.

The rest of the paper is organized in the following way: Section 2 describes some related and similar work. Section 3 gives an introduction to the task-centric programming style and section 4 shows how our run-time system deals with heterogeneity. Sections 5 and 6 give information about the scheduling policies we have developed as well as information concerning the benchmark and the system-under-test. We finish with sections 7 and 8 that shows the experimental results and the conclusion.

2 Related Work

A lot of work have been done to enable heterogeneous multiprocessing. Labarta et al. [3,11,4] introduced Cell/StarSs, a task-centric programming model and run-time system the enables heterogeneous multiprocessing using Cell's seven processors as well as GPUs. Duran et al [5] merged the functionality of StarSs with an OpenMP base to create the OmpSs programming model which contains support for both OpenMP- and StarSs-like syntax. As with StarSs, the OmpSs programming model and its run-time system Nanos++ both support GPUs. Both StarSs and OmpSs are a product of Barcelona Supercomputing Center (BSC). Augonnet et al. introduces the StarPU [1] run-time system that focuses on exploiting multi-heterogeneous systems, primarily GPUs. The programming model concepts are similar to StarSs/OmpSs in that they convey information concerning the tasks memory usage to the run-time system. Augonnet et al.[2] evaluated StarPU using a series of different scheduler with the most popular one being HEFT (Heterogeneous Earliest First) which, similar to our work tries to predict the execution time of tasks. O'Brien et al. [9] evaluated possible support to run OpenMP-annotated programs using the IBM Cell processor. They used similar mechanism as the ones described in this paper, including double buffering and software caching to hide latencies.

Analysis of performance was done using Paraver [10] and speedup figures of up to 16% could be obtained compare to the hand-written benchmark or up to 35x compared to the sequential version when using 8 SPEs. Liu et al. [8] introduces OpenMP extension for heterogeneous SoC architectures. They extended the OpenMP clauses to support their 3SOC architecture thus hiding the complexity of the architecture from the programmer. They evaluated their approach with Matrix Multiplication, SparseLU and FFT and showed linear speedup compared to the sequential version and up-to 30x speedup when using the 3SoC's DSE (Digital Signal Engine). At first glance, the present paper resembles the ideas presented in StarPU [1]. Although there are some small difference, such as the present paper concerns many types of heterogeneous processors perhaps the most notable difference is in the performance model. **In StarPU, the performance model assumes that the execution time is independent from the content of the data** that a task will use. This is not true in a lot of application where the task's data size does not correlate well with the task's complexity (or execution time). Our model uses the information from the user (whatever that may be) to forecast task's with complexity not-seen yet.

3 The Task-Centric Programming Model

The Task-Centric programming model allows a programmer to exploit parallelism within a sequential application by annotating parts of the application source code. Tasks within this programming model should not be confused with other notions of tasks, such as OS tasks or a-prio known tasks or task-graphs. Tasks (and the work within tasks) in the task-centric programming model are dynamic, and they can themselves spawn additional tasks. There is no direct limitation to the complexity of a task; they can range from ten instructions to several millions of instructions. The important property is that tasks can be executed in parallel. Most current task-centric run-time system requires the user to insert task synchronization points in their program. However, there is a way to relax this constraint and let the run-time system itself insert dependencies between tasks; this is done by requiring the programmer to give information about what and how the task will access the memory. Examples of existing run-time systems that supports this implicit way of managing dependencies include the *Ss family (e.g. CellSs[4] and OmpSs[5]). Our run-time system, which is described in section 4 also supports it.

```

1 #pragma omp task input(A) output(D)
   do_work(A,D);
3 #pragma omp task input(D) output(E)
   do_work(D,E);
5 #pragma omp task input(A) output(B)
   do_work(A,B);

```

Fig. 1. OmpSs extended clauses to support automatic dependency insertion transparent to the programmer

Figure 1 shows an OmpSs enabled code which does some work on an array. Tasks are annotated using compiler directive (`#pragma omp` for C) which follows the OpenMP standard. Functions calls or compound statement annotated with the task directive will become task's able to run in parallel. The work is distributed across three tasks that will consume (input) and produce (output) different memory regions. Unlike traditional task-centric models (Cilk, OpenMP,...), an OmpSs supporting run-time system will execute the third task before the second task since the there is no memory dependency between the first (or second) and the third allowing more parallelism to be exposed. This differ from the classical OpenMP approach where a `#pragma omp taskwait` statement would have to be inserted before the last spawned task to ensure consistency.

4 Integration of Heterogeneity in Task-Centric Programming Models

We created a run-time system called UnMP that is capable of supporting the OmpSs programming model. Memory regions specified to be used as input/output/inout are handled by the run-time system which ensures sequential correctness of the parallelized code by preserving data-dependencies between parallel tasks. The run-time system supports i386/x86-64 as a host-device, and GPU(s) or TilePRO64(s) devices as slaves.

Heterogeneity Support

Most of the heterogeneity support within the run-time system is contain within the internal threading system. At the lowest abstraction layers, we are using POSIX-threads (we call them **PHYsical-threads**) to control each of the processing units available at the host system. However, since there is no native or OS control over the external heterogeneous devices, we decided to add another abstract layer called **LOGical-threads**. A LOG-thread is a thread that represents a processor from the run-time systems point of view. For each type of LOG-thread, there is set of API functions that control the selected type. For example, a LOG-thread that represents a GPU will have API functions that starts, finished and prepares a task for the GPU. Using this methodology, we can essentially hide the complexity of using the different units since they look the same from a scheduling point of view. Furthermore, we have support for mapping a LOG-thread to several PHY-threads. This means that we can literally have two different host processors (PHY-threads) controlling the same GPU; should a PHY-thread be busy performing work, another PHY-thread can start-up any work on the GPU improving the resource utilization of the system. The difference between previous work, such as OmpSs[5] and StarPU [1] is also that we focus on adding a third heterogeneous system: TilePRO64. At first glance, it may seem that we use the TilePRO64 as a GPU accelerator. They do bear a resemblance, however, while using a GPU allows access to a primitive bare-metal interface, where the performance relies on the programmer's skill to produce well-behaved code, our implementation open up several **user-transparent optimizations** on the TilePRO64. For example, branch-divergence does not have as big negative impact on performance on the TilePRO64 as it would have on the GPU due to how our TilePRO64 scheduler works. Furthermore, this opens up for a treasure of future work

where we can adopt several locality or energy related techniques on the TilePRO64 side; something that cannot be done on the GPU due to its bare-metal interface.

Amortizing Transfer Costs

One bottleneck of using current heterogeneous devices is that they usually sit on the PCI-express bus which, compared to RAM, has a relatively low bandwidth. To amortize (hide) transfer costs, the UnMP run-time system uses techniques that either reduces or removes overheads related to memory transfers. **Double buffering** enables the run-time system to transfer data associated with the next task to execute while the current task is running. This enables the run-time system to hide the latency associated with upcoming tasks. Another method is to employ a **software cache** that monitors where valid copies exist in our distributed heterogeneous system, as well as properly invalidating the copies when they are overwritten, we can exploit temporal locality found in the application and reduce the memory transfer overhead. The software cache behaves as a SI-cache, with each memory region being either Shared or Invalid, similar to [3,11,4]. Other variations of the protocol includes the MSI protocol employed by StarPU[1].

Writing Heterogeneous Code

The application developer is responsible for creating the different versions of each of the task if heterogeneity support is to be used as well as specifying the task's data usage. Spawning a task with dependencies and with heterogeneous support is conceptually shown in figure 2 using the *device* and *implements* proposed by [5]. The *device* clause specifies the different architectures that the task have been implemented for and *implements* specifies which function this implements. In this example, the task has 3 versions: a host version, a TilePRO64 version and a GPU version. Creating task for the TilePRO64 processor is conceptually very similar to using OpenMP **#pragma omp parallel** directives, enabling SPMD execution. A GPU task should invoke the kernel that implements that particular function. For both the TilePRO64 and GPU case, the memories are maintained and allocated by the run-time system, relieving the programmer from managing them. Figure 2 also show the code-transformation of when a task is spawned. The (#pragma omp task) statement is transformed into a series of library function calls that create a task, sets the memory dependencies and arguments and finally submits the task to be scheduled onto the system resources. We did also include functionality for hinting the run-time system about a task's *complexity*. The *complexity* is a single-value number that can be anything as long as it reflects the parameter that has a profound impact on the task's execution time. Examples of such parameters would be the block-size in matrix multiplication or the amount of rays to cast in a ray tracing program. The complexity clause is used as shown below:

```
#pragma omp task complexity(N)
    matmul_block(A,B,N);
```

The *complexity*, although relatively primitive in its current state have a lot of potential. A smart compiler could theoretically derive the *complexity* itself by analyzing the in-

```

GPU)
2 #pragma omp task device(gpu) implements(inc_arr)
3 void cuda_inc_arr(int *A, int *B)
4 {
5     cuda_inc_array_kernel <<<4,256>>>(A,B);
6 }

```

```

TilePRO64)
2 #pragma omp task device(tilera) implements(inc_arr)
3 void tilera_inc_arr(int *A, int *B)
4 {
5     #pragma omp parallel
6     {
7         int i = omp_get_thread_num();
8         B[i] += A[i];
9     }
10 }

```

```

Task Spawn)
2 #pragma omp task input(A) output(B) target(tilera, gpu, host)
3     inc_arr(&A[0], &B[0]);

```

```

1 Code-Transformation)

3 _unmp_task *tsk = _unmp_create_task();
4 _tsk_arg *arg = (_tsk_arg *) malloc(...);
5 arg->A = &A[0];
6 arg->B = &B[0];
7 _unmp_fan_input (tsk , &A[0] , ...);
8 _unmp_fan_output (tsk , &B[0] , ...);
9 _unmp_set_task_host(tsk , &x86_inc_arr , arg);
10 _unmp_set_task_gpu(tsk , &cuda_inc_arr , arg);
11 _unmp_set_task_tilera(tsk , &tilera_inc_arr , arg);
12 _unmp_submit_task(tsk );

```

Fig. 2. Three different tasks

termediate representation of the task's them self. We consider it future work to improve upon this metric.

5 Heterogeneous Task Scheduling

We developed and implemented the **FCRM (ForeCast-RegressionModel)** scheduler, which is a new scheduling algorithm presented in this paper. We also implemented

three well-known scheduling algorithms in our run-time system to evaluate and compare our FCRM scheduler against: Random, Work-Steal, Weighted-Random. All the four scheduling policies have per-core private task-queues and the major main difference is in the work-dealing strategy.

Random. The *Random* scheduling policy will send task at random to different processing units. It is an architecture- and workload oblivious *work-dealing* algorithm which we use as a baseline due to its familiarity and simplicity.

Work-Steal. The *Work-Steal* is a greedy policy that creates and puts work into the creator's private queue. Load-balancing is achieved by randomly selecting victims to steal work from when the own task-queue is empty. Work-Steal scheduling is a popular policy used in run-time systems such as Wool, Cilk and TBB as well as several OpenMP implementations.

Weighted-Random. The *Weighted-Random* is a work-dealing that distributes work according to each processors weight. The weights were estimated offline using test-runs of the benchmarks on different processing units running in isolation.

FCRM. The FCRM scheduler uses segmented regression to estimate trends concerning the execution time of task on different heterogeneous processors. The idea is to estimate the time a certain task takes on all available processing units and use that information to derive weights according to the estimation. More specifically, we adopt linear regression to fit our data-points (observed task execution time). Should the linear regression fail to fit the entire data-point set to a single linear function, it will segment the data-points so that several function cover the entire fit:

$$f_T(U) = \begin{cases} a_1 + b_1 * U & \text{when } U < BP_1 \\ a_2 + b_2 * U & \text{when } U > BP_1 \text{ and } U < BP_2 \\ \dots \\ a_n + b_n * U & \text{when } U > BP_{n-1} \end{cases}$$

Where BP_n are the calculated break-points for the n segments and U is a task's complexity. For each task, and every processor, we calculate the regressed segmented function. We denote it: $f_{P_T}(U)$ where P is the processor, and T is the task and U is the complexity. To estimate the execution of a task whose complexity has not yet been seen, we have to take where the data is into account:

$$t_{P_{T_U}} = f_{P_T}(U) + g_P(T)$$

where:

$$g_P(T) = BW_{P_{to}} * T_{data-use} + BW_{P_{from}} * T_{data-produce}$$

$BW_{P_{to}}$ = Measured bandwidth to device for processor P

$BW_{P_{from}}$ = Measured bandwidth from device for processor P

$T_{data-use}$ = Data **needed** by task T , taking the software cache into account.

$T_{data-produce}$ = Data **produced** by task T

The FCRM scheduler extends the Weighted-Random scheduling policy by calculating the weight on-line, adapting the any anomalies that were found by the regression. It does also takes the information in the software cache into account when deriving the weights. The entire forecast is recalculated on a miss-prediction (large deviation between predicted and observed time) known after a task have been executed on a particular device; this dynamic re-calculation also allows for adaptation when a certain processing units becomes overloading due to external interference, such as other thermal properties(performance throttling) and other processes sharing that also uses the processor.

6 Experimental Setup

System Specification

We performed the experimental evaluation using a single socket, Quad-Core Intel processor. The processor is connected to a TilePRO64, which is a 64-core chip multiprocessor targeting the embedded market. The system do also contain nVidia Quadro FX 570 CUDA enabled GPU. Both Quadro FX and TilePRO64 sit on the PCI slots.

Benchmark Selection

We selected benchmarks that are well-known and very parallel. This was done intentionally to show that even if the benchmarks themselves are very parallel, when homogeneity stop being a property, even these benchmarks will fail to scale well using well-known scheduling strategies; especially if the tasks are non-uniform concerning the amount of work they perform. In their original form, for each benchmark, the work is divided into segments that can be executed in parallel.

- **Multiple-List-Sort** - *Synthetic* benchmark that simulates incoming packets which contains several lists that needs to be sorted. In the present examples, we assume that the packets are already there, but must be processed one at the time. Each task contains a number of lists, and the lists themselves are of various sizes. The lists are sorted using the Odd-Even sorting mechanism for all architectures. The Odd-Even sort is of $O(n^2)$ complexity and while we understand that the algorithm is not the optimal one for the individual platforms, it is meant to show the benefits and efficiency of various schedulers on this type of application.
- **Matrix Multiplication** - Multiplies two input matrixes A and B to produce matrix C. Matrix multiplication is best suited for GPUs as long as the matrix block size is enough to keep as many of the GPUs threads busy.
- **n-Body** - Simulation of celestial bodies based on classical Newtonian physics. Our implementation is based on the detailed model which $O(n^2)$ in complexity (not the Barnes-Hut algorithm). Tasks are generated to work on subsets of all the celestial bodies; these subsets are of varying length to pronounce the different complexities of tasks.

7 Experimental Results

7.1 Experimental Results

We evaluated our run-time system together with difference scheduling approach under a configuration consisting of three x86-64 processors, one CUDA-enabled GPU and one TilePRO64. For the speedup comparison, the execution time for the schedulers running a certain benchmark were normalized to the serial execution time running on one x86-64 processor. We also included the processor with the best individual performance as a reference point. For each scheduler, and for different parameters, we executed the benchmark ten times taking the median to represent the performance. For the Weighted-Random scheduler, the weights were calculated according to the different processors execution time when running in isolation for one particular set of parameters. For the FCRM scheduler we evaluated both when the scheduler is used without any pre-loaded trend estimation function and when a trend estimation function (FCRM-PRELOAD) have been established from a previous run. Figure 3 shows the performance when executing the MLS (Multiple-List-Sort). We evaluated two cases where the total amount of sorted elements varied. For both the cases, we notice that the FCRM-PRELOAD scheduler that uses a trend-regression function from a previous runs performs much better than the other schedulers. The Weighted-Random scheduler performs slightly worse than the Work-Steal in this case. The reason Weight-Random scheduler and the Random scheduler do not perform well is due to the pushing of work to the GPU, which is the slow processor in benchmark.

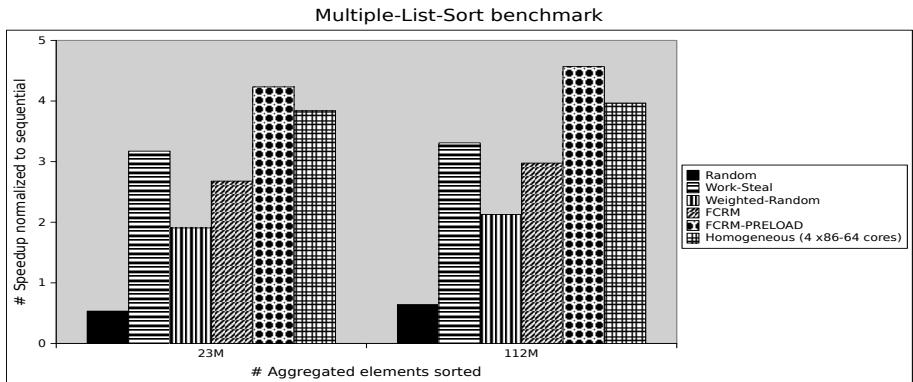


Fig. 3. Multiple-List-Sort benchmark performance and speedup relative sequential execution time. Four x86-64 cores showed as reference point.

Figure 4 shows the performance of the schedulers when running a blocked matrix multiplication. There is only a slight difference between Weighted-Random scheduling policy and the FCRM scheduler that is using an already established estimation function and the different is primarily in that the FCRM also takes the locality of the tasks into account when deciding upon the weight; something that the Weighted-Random scheduling policy does not. Also notice that even for a uniform benchmark such as matrix multiplication neither the Random nor Work-Steal perform any good. The reason

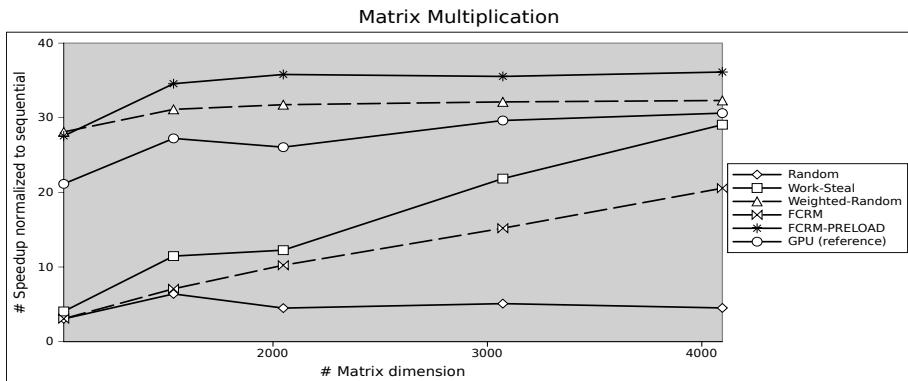


Fig. 4. Matrix multiplication speedup amongst schedulers with the GPU as a reference point

for this due to the work being stolen by the TilePRO64 thread as soon as it is idle, and the TilePRO64 is the slowest processor to execute the matrix multiplication code in this benchmark.

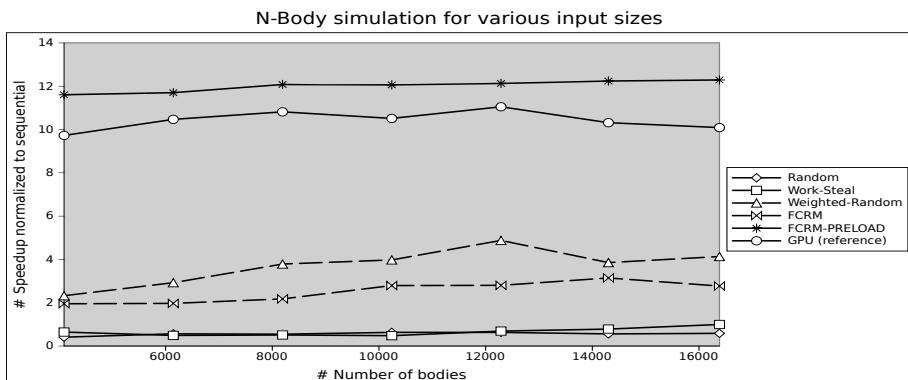


Fig. 5. N-body speed-up for 30 time-step iterations with GPU as a reference point

Figure 5 shows the performance of the schedulers when running the simulation of N-Body. Our implementation of N-Body is highly non-uniform, making the weights calculated for the Weight-Random scheduler near useless as it does not take the complexity of tasks into account. For example, a task that calculates the gravitational force of 10 bodies have equally high chance of being scheduled on the TilePRO64 as a task that computes 1000 bodies; even though the latter should instead be placed onto the GPU to reduce the critical path. The FCRM scheduler without an initial estimation function performs slightly worse than Weighted-Random. This is one of the drawbacks on using the FCRM scheduler without an initial estimate function on small applications (with few iterations).

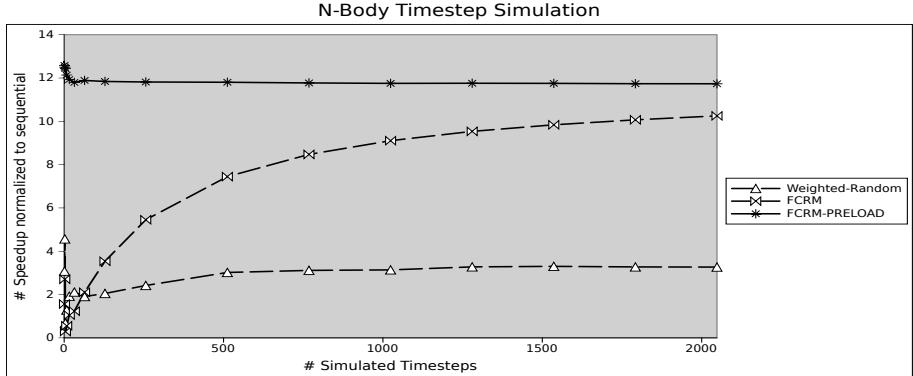


Fig. 6. Impact of application length on FCRM with and without preloaded regression function

However, when we increase the duration (figure 6) of the application, we see that the FCRM scheduler without the initial estimate function slowly converges to the result of the scheduler with an initial estimate function.

8 Conclusions

In the present paper, we showed how to utilize heterogeneity that differs in several aspects: frequency, core count and core type. We presented a task-centric run-time system capable of handling the OmpSs programming model as well as extensions for improving the transfers by utilizing software caching and overlapped memories. We further gave example of a scheduler capable of both adapting-to and improving the execution time of a series of benchmark; benchmarks that are highly dynamic in nature. This dynamic nature makes scheduling a large workload on a bad processor very inefficient, which we have captured by extending the Weighted-Random scheduling approach with trend analysis for weight estimation. We have shown that our scheduler performs significantly better than the original Weighted-Random, and far better than Random Work-Stealing approach which is a well-known and embraced scheduling policy for homogeneous cores and workloads.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the ENCORE Project (www.entre-project.eu), grant agreement nr. 248647. The authors are members of the HiPEAC European network of Excellence (<http://www.hipeac.net>). We would like to thank Alejandro Rico and the team at BSC for reference designs that uses the data-driven task-extensions of OmpSs.

References

1. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.-A.: STARPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 863–874. Springer, Heidelberg (2009)
2. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.: Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience* 23(2), 187–198 (2011)
3. Ayguadé, E., Badia, R.M., Igual, F.D., Labarta, J., Mayo, R., Quintana-Ortí, E.S.: An Extension of the StarSS Programming Model for Platforms with Multiple GPUs. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 851–862. Springer, Heidelberg (2009)
4. Bellens, P., Perez, J., Badia, R., Labarta, J.: Cellss: a programming model for the cell be architecture. In: SC 2006 Conference, Proceedings of the ACM/IEEE, p. 5. IEEE (2006)
5. Duran, A., Ayguadé, E., Badia, R., Labarta, J., Martinell, L., Martorell, X., Planas, J.: Ompss: A proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters* 21(2), 173–193 (2011)
6. Faxén, K.: Wool-a work stealing library. *ACM SIGARCH Computer Architecture News* 36(5), 93–100 (2009)
7. Frigo, M., Leiserson, C., Randall, K.: The implementation of the cilk-5 multithreaded language. *ACM Sigplan Notices* 33(5), 212–223 (1998)
8. Liu, F., Chaudhary, V.: Extending openmp for heterogeneous chip multiprocessors. In: Proceedings of the 2003 International Conference on Parallel Processing, pp. 161–168. IEEE (2003)
9. O’Brien, K., O’Brien, K., Sura, Z., Chen, T., Zhang, T.: Supporting openmp on cell. *International Journal of Parallel Programming* 36(3), 289–311 (2008)
10. Pillet, V., Labarta, J., Cortes, T., Girona, S.: Paraver: A tool to visualise and analyze parallel code. In: Proceedings of WoTUG-18: Transputer and occam Developments, vol. 44, pp. 17–31 (1995)
11. Planas, J., Badia, R., Ayguadé, E., Labarta, J.: Hierarchical task-based programming with starss. *International Journal of High Performance Computing Applications* 23(3), 284–299 (2009)

Weighted Block-Asynchronous Iteration on GPU-Accelerated Systems

Hartwig Anzt¹, Stanimire Tomov², Jack Dongarra^{2,3,4}, and Vincent Heuveline¹

¹ Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

² University of Tennessee, Knoxville, USA

³ Oak Ridge National Laboratory, Oak Ridge, USA

⁴ University of Manchester, Manchester, UK

{hartwig.anzt,vincent.heuveline}@kit.edu,
{tomov,dongarra}@cs.utk.edu

Abstract. In this paper, we analyze the potential of using weights for block-asynchronous relaxation methods on GPUs. For this purpose, we introduce different weighting techniques similar to those applied in block-smoothers for multigrid methods. For test matrices taken from the University of Florida Matrix Collection we report the convergence behavior and the total runtime for the different techniques. Analyzing the results, we observe that using weights may accelerate the convergence rate of block-asynchronous iteration considerably. While component-wise relaxation methods are seldom directly applied to systems of linear equations, using them as smoother in a multigrid framework they often provide an important contribution to finite element solvers. Since the parallelization potential of the classical smoothers like SOR and Gauss-Seidel is usually very limited, replacing them by weighted block-asynchronous smoothers may be beneficial to the overall multigrid performance. Due to the increase of heterogeneity in today's architecture designs, the significance and the need for highly parallel asynchronous smoothers is expected to grow.

Keywords: asynchronous relaxation, weighted block-asynchronous iteration methods, multigrid smoother, GPU.

1 Introduction

Using weights in iterative relaxation schemes is a well known and often applied technique to improve the convergence. While the classical successive over-relaxation method (SOR, [Saa03]) consists of a weighted Gauss-Seidel, especially the block smoothers in multigrid methods are often enhanced with weights to improve their convergence [BFKMY11]. In these the parallelized Block-Jacobi- or Block-Gauss-Seidel smoothers are weighted according to the block decomposition of the matrix. In [ATG⁺11] we explored the potential of replacing the classically applied smoothers in multigrid methods by asynchronous iterations. While a block parallelized smoother requires synchronization between the iterations, asynchronous methods are very tolerant to component update order and

latencies concerning the communication of updated component values. This lack of synchronization barriers makes them suitable candidates for modern hardware architectures, often accelerated by highly parallel coprocessors. In [ATDH11] we have shown how to enhance asynchronous iteration schemes to compensate for the inferior convergence rate of the plain asynchronous iteration. In particular, this is achieved by adding local iterations on subdomains that fit into the accelerators' cache, therefore almost come for free, and should not be counted as global iterations. Furthermore, the higher iteration number per time frame on the GPUs potentially results in significant performance increase. While Chazan and Miranker have introduced a weighted asynchronous iteration similar to SOR [CM69], it becomes of interest whether the block-asynchronous iteration benefits from weighting methods similar to those applied to block smoothers [BFKMY11]. The motivation is that in the local iterations performed on the subdomains, the off-block matrix entries are neglected. To account for this issue it may be beneficial to weight the local iterations. This can be achieved either by using ℓ_1 -weights, by a technique similar to ω -weighting in SOR, or by a combination of both. The purpose of this paper is to introduce the different methods and report experimental results on the convergence rate as well as the time-to-solution performance.

2 Mathematical Background

2.1 Asynchronous Iteration

The Jacobi method is an iterative algorithm for finding the approximate solution to a linear system of equations $Ax = b$ that converges if A is strictly or irreducibly diagonally dominant [Var10], [Bag95]. One can decompose the system into $(L + D + U)x = b$ where D denotes the diagonal entries of A while L and U denote the lower and upper triangular part of A , respectively. Using the form $Dx = b - (L + U)x$, the Jacobi method is derived as an iterative scheme where the matrix $B \equiv I - D^{-1}A$ is often referred to as iteration matrix. It can also be rewritten in the following component-wise form:

$$x_i^{m+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j^m \right) = \sum_{i=1}^n B_{ij} x_j^m + d_i \quad (1)$$

where $b_{ij} = (B)_{ij}$ with $B = I - D^{-1}A$ and $d_i = \frac{b_i}{a_{ii}}$ for all $i, j = 1 \dots n$. For computing the next iteration, one needs the latest values of all components. This requires a strict order of the component updates, limiting the parallelization potential to a stage, where no component can be updated multiple times before all the other components are updated. If this order is not adhered, i.e. the individual components are updated independently and without consideration of the current state of the other components, the resulting algorithm is called chaotic or asynchronous iteration method [FS00]. Back in the 70's Chazan and Miranker analyzed some basic properties of these methods, and established

convergence theory [CM69], [Str97] [AD86], [BE86]. For the last 30 years, these algorithms came out of focus of high-performance computing due to the superior performance of synchronized iteration methods. More interest was put on the convergence properties and deriving models for the computational cost [AD89], [Bah97], [BSS99], [DB91]. Today, due to the complexity of heterogeneous hardware platforms and the high number of computing units in parallel devices like GPUs, these schemes may become interesting again for applications like multi-grid methods, where highly parallel smoothers are required on the distinct grid levels. While traditional smoothers like the sequential Gauss-Seidel obtain their efficiency from their fast convergence, it may be true that the asynchronous iteration scheme overcompensate the inferior convergence behavior by superior scalability [ATG⁺11].

2.2 Block-Asynchronous Iteration

One possible motivation for the block-asynchronous iteration comes from the hardware architecture. The idea is to split the linear system into blocks of rows, and then to assign the computations for each block to one thread block on a graphics processing unit (GPU). For these thread blocks, an asynchronous iteration method is used, while on each thread block, instead of one, multiple Jacobi-like iterations are performed. During these local iterations on the matrix block, the x values used from outside the block are kept constant (equal to their values at the beginning of the local iterations). After the local iterations, the updated values are communicated. In other words, using domain-decomposition terminology, the blocks correspond to subdomains and thus the method iterates locally on every subdomain. We denote this scheme by *async-(local_iters)*, where the index *local_iters* indicates the number of Jacobi-like updates on each subdomain [ATDH11]. As the subdomains are relatively small and the data needed for the local iterations largely fits into the multiprocessor's cache, these additional iterations on the subdomains almost come for free. This approach, inspired by the well known hybrid relaxation schemes [BFKMY11], [BFG⁺], arises as a specific case of an asynchronous two-stage method [BMPS99].

The obtained algorithm can be written as component-wise update of the solution approximation:

$$x_k^{(m+1)+} = \frac{1}{a_{kk}} \left(b_k - \underbrace{\sum_{j=1}^{T_S-1} a_{kj} x_j^{(m-\nu(m+1,j))}}_{\text{global part}} - \underbrace{\sum_{j=T_S}^{T_E} a_{kj} x_j^{(m)}}_{\text{local part}} - \underbrace{\sum_{j=T_E+1}^n a_{kj} x_j^{(m-\nu(m+1,j))}}_{\text{global part}} \right)$$

where T_S and T_E denote the starting and the ending indices of the matrix/vector part in the thread block. In the component updates on the subdomains, for the local components, the most recent values are used, while for the global part, the values from the beginning of the iteration are used. The shift function $\nu(m+1, j)$ denotes the iteration shift for the component j which can be positive or negative,

depending on whether the thread block where the component j is located in already has conducted more or less iterations. Note that this may give a block Gauss-Seidel flavor to the updates.

2.3 Weights in Block-Asynchronous Iteration

To examine the topic of weights in the block-asynchronous iteration, we introduce some notation to simplify the analysis [BFKMY11]. Splitting the matrix A into blocks, we use A_{kk} for the matrix block located in the k -th block row and the k -th block column. Furthermore, we introduce the index sets Ω_k , where

$$\Omega = \bigcup_{k=1}^p \Omega_k = \{1, 2 \dots n\},$$

and $\Omega_i \cap \Omega_j = \emptyset \forall i \neq j$ consistent to the block decomposition of the matrix A . Using this notation, Ω_k contains all indices j with $T_S(k) \leq j \leq T_E(k)$ where $T_S(k)$ (respectively $T_E(k)$) denotes the first (last) row and column index of the diagonal block A_{kk} . We now define the sets

$$\Omega^{(i)} = \{j \in \Omega_k : i \in \Omega_k\}, \quad \Omega_0^{(i)} = \{j \notin \Omega_k : i \in \Omega_k\}.$$

Hence, for block A_{kk} , $\Omega^{(i)}$ contains the indices with corresponding columns being part of the diagonal block of row i while $\Omega_0^{(i)}$ contains the indices of the columns that have no entries in the block. This way, we can decompose the sum of the elements of the i -th row:

$$\sum_j a_{ij} = \underbrace{\sum_{j=1}^{T_S-1} a_{ij}}_{\text{off-block columns}} + \underbrace{\sum_{j=T_S}^{T_E} a_{ij}}_{\text{block columns}} + \underbrace{\sum_{j=T_E+1}^n a_{ij}}_{\text{off-block columns}} = \underbrace{\sum_{j \in \Omega^{(i)}} a_{ij}}_{\text{block columns}} + \underbrace{\sum_{j \in \Omega_0^{(i)}} a_{ij}}_{\text{off-block columns}}. \quad (2)$$

Similar to the ω -weighted asynchronous iteration [CM69], it is possible to use ω -weights for the block structure in the block-asynchronous approach. In this case, the solution approximation of the local iterations is weighted when updating the global iteration values. The parallel algorithm for the component updates in one matrix block is outlined in Algorithm 1.

Beside this general weighting method, we introduce a more sophisticated technique, that is usually referred to as ℓ_1 -weighting [BFKMY11].

Classically applied to Block-Jacobi and Gauss-Seidel relaxation methods, ℓ_1 weighting modifies the iteration matrix by replacing $B = I - D^{-1}A$ with $B_{\ell_1} = I - (D + D^{\ell_1})^{-1}A$, where D^{ℓ_1} is the diagonal matrix with entries

$$d_{ii}^{\ell_1} := \mathbf{sign}(a_{ii}) \sum_{j \in \Omega_0^{(i)}} |a_{ij}|. \quad (3)$$

- 1: Update component i :
- 2: $s := d_i + \sum_{j \in \Omega_0^{(i)}} b_{ij}x_j$ {off-diagonal part}
- 3: $x_i^{local} = x$
- 4: **for all** $k = 0; k < local_iters; k++$ **do**
- 5: $x_i^{local} := s + \sum_{j \in \Omega^{(i)}} b_{ij}x_j^{local}$ {using the local updates in the block}
- 6: **end for**
- 7: $x_i = \omega x_i^{local} + (1 - \omega)x_i$

Algorithm 1. Basic principle of using ω weights in block-asynchronous iteration featuring $local_iters$ local iterations

Table 1. Dimension and characteristics of the SPD test matrices and the corresponding iteration matrices

Matrix name	#n	#nnz	con(A)	con($D^{-1}A$)	$\rho(M)$
CHEM97ZTZ	2,541	7,361	1.3e+03	7.2e+03	0.7889
FV1	9,604	85,264	9.3e+04	12.76	0.8541
FV3	9,801	87,025	3.6e+07	4.4e+03	0.9993
TREFETHEN_2000	2,000	41,906	5.1e+04	6.1579	0.8601

The advantage over the across-the-board ω -weighting technique is that it applies different weights in the distinct rows, which accounts for the respective off-diagonal entries.

3 Numerical Experiments

In our experiments, we search for the approximate solutions of linear system of equations, where the respective matrices are taken from the University of Florida Matrix Collection (UFMC; see <http://www.cise.ufl.edu/research/sparse/matrices/>). Due to the convergence properties of the iterative methods we analyze, the experiment matrices have to be chosen properly, fulfilling the sufficient convergence condition [Str97]. The matrix properties and sparsity plots can be found in Table 1 and Figure 1. We furthermore take set the right-hand side in $Ax = b$ to $b \equiv 1$ for all linear systems.

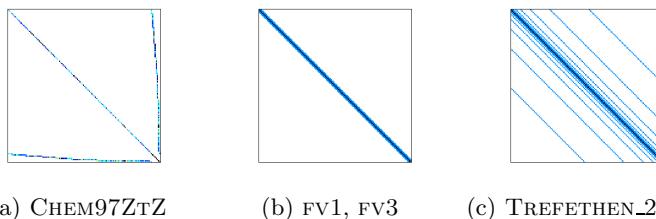


Fig. 1. Sparsity plots of test matrices

The experiments were conducted on a heterogeneous GPU-accelerated multicore system located at the University of Tennessee, Knoxville. The system's

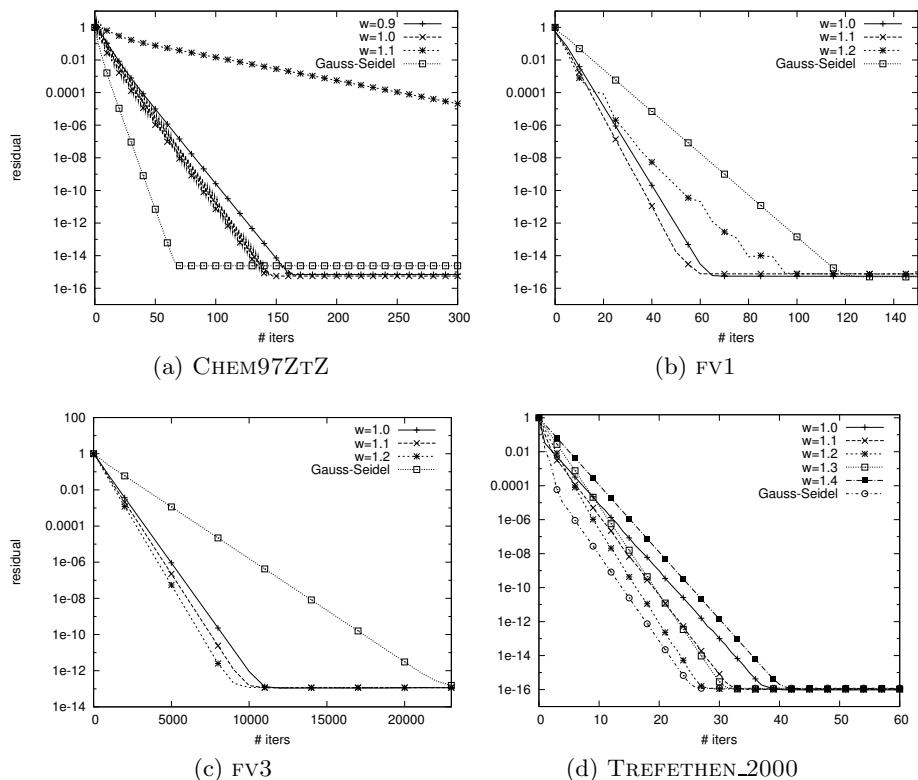


Fig. 2. Convergence rate of ω -weighted block-asynchronous iteration compared to Gauss-Seidel convergence. The (relative) residuals are always in L^2 norm.

CPU is one socket Intel Core Quad Q9300 @ 2.50GHz and the GPU is a Fermi C2050 (14 Multiprocessors x 32 CUDA cores @1.15GHz, 3 GB memory). The GPU is connected to the CPU host through a PCI-e \times 16. On the CPU, the synchronous Gauss-Seidel and SOR implementations run on 4 cores. The Intel compiler 11.1.069 [int] is used with optimization flag “-O3”. In the GPU implementation, based on CUDA [NVI09] with the respective libraries taken from CUDA 4.0.17 [NVI11], the component updates use a thread blocks of size 512. (Except for the ℓ_1 weighted method, where the thread block size is chosen consistent to the matrix block size.) The thread block size, the number of streams, along with other parameters, were determined through empirically based tuning.

In a first experiment, we analyze the influence of ω -weighting on the convergence rate with respect to global iteration numbers. For this purpose we plot the relative residual depending on the iteration number. Note that all values are average due to the non-deterministic properties of block-asynchronous iteration. To have a reference, we additionally provide in Figure 2 the convergence behavior of the sequential Gauss-Seidel algorithm. The results reveal that the convergence rate of the block-asynchronous iteration is very dependent on the

matrix characteristics. For matrices with most relevant matrix entries gathered on or near the diagonal, the local iterations provide sufficient improvement to compensate for the inferior convergence rate of the asynchronous iteration conducted globally. In these cases, e.g. FV1 and FV3, we achieve a higher convergence rate than the sequential Gauss-Seidel algorithm. Similar to the SOR algorithm, choosing $\omega > 1$ may even improve the convergence for specific problems (Figure 2b and 2c). For systems with considerable off-diagonal entries, the convergence of the block-asynchronous iteration decreases considerably compared to the Gauss-Seidel scheme (Figure 2a, 2d). The reason is, that the off-diagonal entries are ignored in the local iterations.

While the ω -technique applies a general weighting to account for the off-diagonal entries, the more sophisticated ℓ_1 -weighting technique applies different weights in different rows. To analyze the impact of ℓ_1 -weighting we focus on the matrix TREFETHEN_2000 where the ratio between the entries in the diagonal block and the off-diagonal parts differs significantly for the distinct rows.

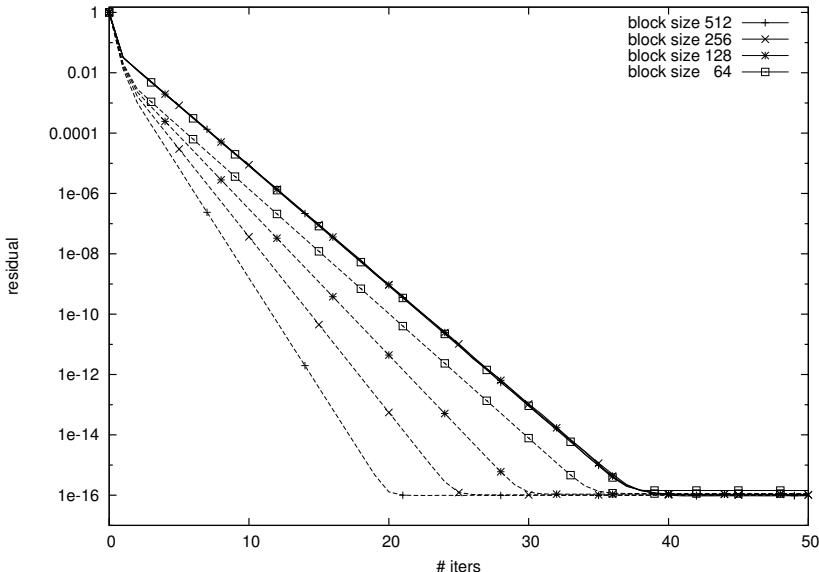


Fig. 3. Convergence improvement using ℓ_1 -weights applied to TREFETHEN_2000 for different block sizes. Solid lines, all lying on top of each other, are block-asynchronous iteration, dashed lines are block-asynchronous iteration using ℓ_1 -weights. The (relative) residuals are always in L^2 norm.

We can observe in Figure 3 that, independently of the block size, using ℓ_1 weights improves the convergence rate. We also note that the influence of the block-size on the convergence rate for the unweighted algorithm is negligible. Furthermore, using ℓ_1 weights is especially beneficial when targeting large block sizes, where the ℓ_1 weights for the distinct rows in one block differ considerably. For this case (e.g. block size 512), the convergence of the block-asynchronous

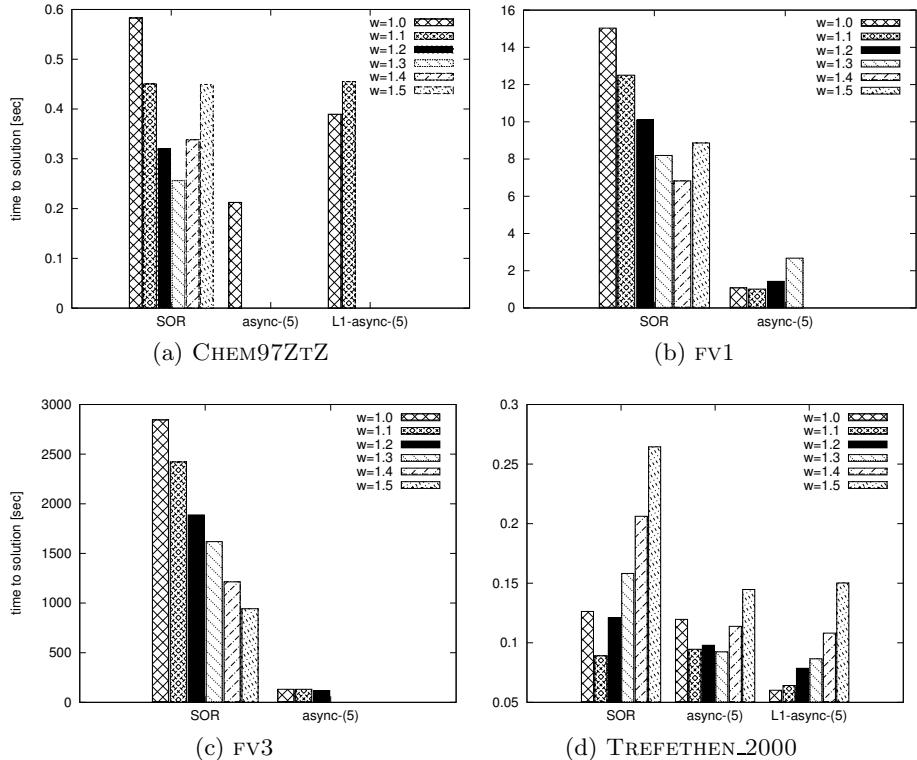


Fig. 4. Time-to-solution comparison between SOR and ω -weighted block-asynchronous iteration (async-(5)). In (4a) and (4d) we additionally provide the data for the combination of ω - and ℓ_1 -weighting (L1-async-(5)).

iteration is improved by a factor of almost 2 compared to the unweighted algorithm.

While the convergence rate, with respect to iteration number, is interesting from the theoretical point of view, the more relevant factor is the time-to-solution performance. This depends not only on the convergence rate, but also on the efficiency of the respective algorithm on the available hardware resources (hardware-dependent iteration rate). Whereas the Gauss-Seidel algorithm and the derived SOR algorithms require strict update order and hence only allow sequential implementations, block-asynchronous iteration is very tolerant to update order and synchronization latencies, and therefore adequate for GPU implementations. In the next experiment, we analyze the time to solution for the ω -weighted block-asynchronous iteration and compare it with the SOR algorithm. We want to stress that despite the similar notation, ω -weighting has, due to the algorithm design, a very different meaning in the SOR and the block-asynchronous iter-

tion, respectively. While ω weights in SOR the individual iterations, in $\text{async-}(5)$ it is used to weight the local iterations with respect to the global ones.

For the matrices with considerable off-diagonal entries (large d_{ii} in (3)), we provide additional data for different ω -weights applied to the block-asynchronous algorithm enhanced by the ℓ_1 -weighting technique. The results show that for matrices where most entries are clustered on or near the main diagonal, the ω -weighted block asynchronous iteration outperforms the SOR method by more than an order of magnitude, see Figure 4b and 4c. But at the same time, ω -weights for the block-asynchronous algorithm have to be applied more carefully: already choosing $\omega \geq 1.4$ leads to divergence of all test cases. For matrices with considerable off-diagonal parts, using the block-asynchronous iteration may not pay off when comparing with SOR. Considering the runtime analysis for the matrix CHEM97ZTZ (Figure 4a) we have to realize that although the unweighted block-asynchronous iteration generates the solution faster than SOR, using ω -weights is not beneficial. The algorithm also does not benefit from enhancing it by ℓ_1 -weights, which may stem from the very unique matrix properties. We notice however that, despite the poor performance, ℓ_1 -weights have positive impact on the algorithm's stability: for $\omega = 1.1$, the convergence of the block-asynchronous iteration is maintained. For the test matrix TREFETHEN_2000, the performance of SOR and block-asynchronous iteration is comparable for ω near 1. But enhancing the latter one with ℓ_1 weights causes significant performance increase for $\text{async-}(5)$. We then outperform the SOR algorithm by a factor of nearly 5 (see Figure 4d). This not only reveals that using ℓ_1 -weights is beneficial to the method's performance, but also the potential of applying a combination of both weighting techniques.

4 Conclusion

We introduced two weighting techniques for block-asynchronous iteration methods that improve the convergence properties. In numerical experiments with linear systems of equations taken from the University of Florida Matrix collection we were able to show how the different techniques improve not only the convergence rate but also the time-to-solution performance. The further research in this field will focus on how these improvements by using weights in block-asynchronous methods transfer to multigrid methods. Especially algebraic multigrid, where considerable off-diagonal entries are expected on the different grid levels, may benefit from using weighted block-asynchronous iteration smoothers.

References

- [AD86] Aydin, U., Dubois, M.: Generalized asynchronous iterations, pp. 272–278 (1986)
- [AD89] Aydin, U., Dubois, M.: Sufficient conditions for the convergence of asynchronous iterations. *Parallel Computing* 10(1), 83–92 (1989)

- [ATDH11] Anzt, H., Tomov, S., Dongarra, J., Heuveline, V.: A block-asynchronous relaxation method for graphics processing units. Technical report, Innovative Computing Laboratory, University of Tennessee, UT-CS-11-687 (2011)
- [ATG⁺11] Anzt, H., Tomov, S., Gates, M., Dongarra, J., Heuveline, V.: Block-asynchronous Multigrid Smoothers for GPU-accelerated Systems. Technical report, Innovative Computing Laboratory, University of Tennessee, UT-CS-11-689 (2011)
- [Bag95] Bagnara, R.: A unified proof for the convergence of jacobi and gauss-seidel methods. *SIAM Rev.* 37, 93–97 (1995)
- [Bah97] Miellou, J.C., Rhofir, K., Bahi, J.: Asynchronous multisplitting methods for nonlinear fixed point problems. *Numerical Algorithms* 15(3-4), 315–345 (1997), cited By (since 1996) 23
- [BE86] Bertsekas, D.P., Eckstein, J.: Distributed asynchronous relaxation methods for linear network flow problems. In: *Proceedings of IFAC 1987* (1986)
- [BFG⁺] Baker, A.H., Falgout, R.D., Gamblin, T., Kolev, T.V., Martin, S., Yang, U.M.: Scaling algebraic multigrid solvers: On the road to exascale. In: *Proceedings of Competence in High Performance Computing, CiHPC 2010* (2010)
- [BFKMY11] Baker, A.H., Falgout, R.D., Kolev, T.V., Yang, U.M.: Multigrid smoothers for ultra-parallel computing, LLNL-JRNL-435315 (2011)
- [BMPS99] Bai, Z.-Z., Migallón, V., Penadés, J., Szyld, D.B.: Block and asynchronous two-stage methods for mildly nonlinear systems. *Numerische Mathematik* 82, 1–20 (1999)
- [BSS99] Blathras, K., Szyld, D.B., Shi, Y.: Timing models and local stopping criteria for asynchronous iterative algorithms. *Journal of Parallel and Distributed Computing* 58, 446–465 (1999)
- [CM69] Chazan, D., Miranker, W.: Chaotic Relaxation. *Linear Algebra and Its Applications* 2(7), 199–222 (1969)
- [DB91] Dubois, M., Briggs, F.A.: The run-time efficiency of parallel asynchronous algorithms. *IEEE Trans. Computers* 40(11), 1260–1266 (1991)
- [FS00] Frommer, A., Szyld, D.B.: On asynchronous iterations. *Journal of Computational and Applied Mathematics* 123, 201–216 (2000)
- [int] Intel C++ Compiler Options. Intel Corporation. Document Number: 307776-002US
- [NVI09] NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture Programming Guide, 2.3.1 edn. (August 2009)
- [NVI11] NVIDIA Corporation. CUDA TOOLKIT 4.0 READINESS FOR CUDA APPLICATIONS, 4.0 edn. (March 2011)
- [Saa03] Saad, Y.: Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia (2003)
- [Str97] Strikwerda, J.C.: A convergence theorem for chaotic asynchronous relaxation. *Linear Algebra and its Applications* 253(1-3), 15–24 (1997)
- [Var10] Varga, R.S.: Matrix Iterative Analysis. Springer Series in Computational Mathematics. Springer (2010)

An Optimized Parallel IDCT on Graphics Processing Units

Biao Wang¹, Mauricio Alvarez-Mesa^{1,2}, Chi Ching Chi¹, and Ben Juurlink¹

¹ Embedded Systems Architecture, Technische Universität Berlin, Berlin, Germany
biaowang@mailbox.tu-berlin.de,
{mauricio.alvarezmesa,cchi,b.juurlink}@tu-berlin.de
<http://www.aes.tu-berlin.de/>

² Multimedia Communications, Fraunhofer HHI, Berlin, Germany

Abstract. In this paper we present an implementation of the H.264/AVC Inverse Discrete Cosine Transform (IDCT) optimized for Graphics Processing Units (GPUs) using OpenCL. By exploiting that most of the input data of the IDCT for real videos are zero valued coefficients a new compacted data representation is created that allows for several optimizations. Experimental evaluations conducted on different GPUs show average speedups from $1.7\times$ to $7.4\times$ compared to an optimized single-threaded SIMD CPU version.

Keywords: IDCT, GPU, H.264, OpenCL, parallel programming.

1 Introduction

Currently H.264/AVC is one of the most widely used video codecs in the world [1]. It achieves significant improvements in coding performance compared to previous video codecs at the cost of higher computational complexity. Single-threaded performance, however, is no longer increasing at the same rate and now performance scalability is determined by the ability of applications to exploit thread-level parallelism on parallel architectures.

Among different parallel processors available today Graphics Processing Units (GPUs) have become popular for general-purpose computing because of its high computational capabilities and the availability of general purpose GPU programming models such as CUDA [2] and OpenCL [3].

GPUs can accelerate applications to a great extent as long as they feature massive and regular parallelism. Video decoding applications, however, do not meet these requirements completely because of different block sizes and multiple prediction modes.

H.264 decoding consists of 2 main stages, namely: entropy decoding and macroblock reconstruction. The latter, in turn, includes the inverse transform, coefficient rescaling, intra- and inter-prediction and the deblocking filter. Among them, the inverse transform is a good candidate for GPU acceleration because it has only two block sizes and the blocks in frame can be processed independently.

The H.264 transform is an integer approximation of the well known Discrete Cosine Transform (DCT) [4]. The main transform is applied to 4×4 blocks and, the H.264 High Profile, currently the most widely used profile, allows another transform for 8×8 blocks [5].

In H.264 High Profile (that uses the 4:2:0 color format) each picture is divided in macroblocks, each one consisting of one block of 16×16 luma samples and two blocks of 8×8 chroma samples. For the luma component the encoder is allowed to select between the 4×4 and 8×8 transforms on a macroblock by macroblock basis. For the chroma components only the 4×4 transform is allowed. The general form of the Inverse DCT (IDCT), that is applied in the H.264 decoder, is defined in Equation 1:

$$\mathbf{X} = \mathbf{C}^T \mathbf{Y} \mathbf{C} \quad (1)$$

where Y is a matrix of input coefficients, X is a matrix of output residual data and C is the transform matrix. In H.264, C is defined in such a way that the transform can be performed only using integer operations without multiplications. Typically, the IDCT is not implemented using a matrix multiplication algorithm. Instead, the 2D-IDCT is implemented as two 1D-IDCT using a row-column decomposition [6]. Figure 1 shows a flow diagram of 4×4 1D-IDCT.

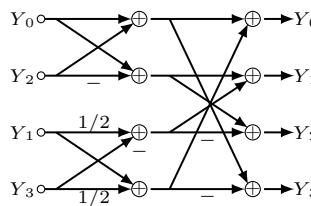


Fig. 1. 4×4 1D-IDCT

In this paper we present an H.264 IDCT implementation optimized for GPUs using OpenCL with portability purpose. We exploit the fact that a significant part of the input data consists of zero data to create an efficient data representation. This simplifies the computation on the GPU and reduces the amount of data that has to be transferred between CPU and GPU memories. On the GPU kernel itself, additional optimizations are applied such as synchronization removal, workgroup enlargement, data granularity enlargement and coalesced write back.

Some previous work has parallelized the IDCT on GPUs. Fang et. al. implemented an 8×8 JPEG IDCT using single precision floating point arithmetic on GPUs using Microsoft DirectX9.0 [7]. The paper shows that their optimal GPU kernel is faster than a CPU optimized kernel with MMX (64-bit SIMD) but slower than SSE2 (128-bit SIMD).

As part of the NVIDIA CUDA Software Development Kit there are two implementations of the JPEG 8×8 DCT/IDCT, one using single precision floating

point and the other one using 16-bit integer arithmetic [8]. However, no performance analysis or a comparison with CPU optimized codes is performed. The algorithm presented in this paper uses a similar thread mapping strategy but with some enhancements for improving the data access efficiency. Furthermore, the IDCT in H.264 is more complex as it contains two different transforms: 4×4 and 8×8 .

The rest of paper is organized as follows: The parallelization strategy and the main optimization techniques are presented in Section 2. Experimental setup and results are presented in Section 3. Finally, in Section 4 conclusions are drawn.

2 Implementation of IDCT on GPU

Our GPU implementation is based on an optimized CPU version of the H.264 decoder that, in turn, is based on FFmpeg [9]. In our base code, entropy decoding and macroblock reconstruction have been decoupled into several frames passes. In order to offload the IDCT kernel to the GPU, we further decouple the IDCT from the macroblock reconstruction loop, and create a separated frame pass for it as well. Decoupling requires intermediate frame buffers for input and output data.

The general concept for offloading the IDCT to the GPU is as follows. First the CPU performs entropy decoding on the whole frame and produces the input buffer for the IDCT. This buffer is transferred from the CPU to the GPU, termed as host and device in OpenCL, respectively. Then, the GPU performs the IDCT for the whole frame. When finished, the results are transferred back to the host memory. Finally, the CPU performs the remaining macroblock reconstruction stages.

2.1 Compaction and Separation

A baseline computation of GPU IDCT includes detecting the type of IDCT 4×4 and 8×8 , as well as detecting and skipping blocks with zero coefficients. However, skipping zero blocks has no benefit when it results in branch divergence.

We investigate the input of IDCT kernel. Figure 2 shows the average non-zero blocks ratio for sequences in different resolutions (1080p and 2160p) and 3 different encoding modes (labeled as CRF37, CRF22, and Intra). The details of the video sequences and encoding modes are presented in Section 3. According to the Figure 2, a considerable portion of the input are zero blocks.

To avoid the computation and memory transfer for the non-zero blocks, we propose two preprocessing steps done by CPU based on the high ratio of zero blocks within input, referred as compaction and separation. With compaction, all zero blocks are squeezed out before they are transferred to GPU, relieving memory transfer overhead to a great extent. With separation, input data are separated into two different buffers according to their IDCT type.

In order to reduce the overhead of these preprocessing steps, the entropy decoding stage has been adapted for producing dense buffers, separated for 4×4 and 8×8 blocks. Introducing these steps has no noticeable effect in the execution time.

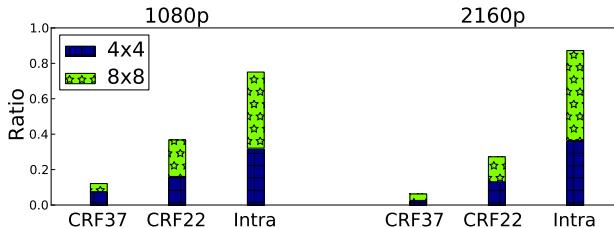


Fig. 2. Non-zero blocks ratio for different encoding modes

2.2 GPU Kernel Implementation

The computation of the IDCT is performed in an OpenCL kernel, a function executed on the GPU. OpenCL uses a hierarchical data-parallel programming model with two-levels. At the higher level, the global thread index space, termed as *NDRange*, is divided into *workgroups*. At the lower level each workgroup is further divided into *workitems*. Each workitem is an instance of a kernel execution, i.e a thread [3].

For the IDCT, three main options are possible for assigning data elements to threads: sample, line and block. Sample to thread mapping, in which each output sample computation is assigned to a thread, leads to divergence as calculation for each sample is different. Line to thread mapping is chosen over block to thread mapping for our implementation because it supplies a larger amount of parallelism. Block to thread mapping, in which each 4×4 or 8×8 block computation is assigned to a thread, leads to insufficient parallelism. Then, line to thread mapping, in which each thread processes one line (row and then column) of coefficients, represents a good solution and it is chosen for our implementation.

In the baseline kernel, the size of workgroup is chosen as 96 threads to match the size of macroblock. For better comparison, the compacted and separated kernel has the same size of workgroup, with a configuration of 32 and 3 in dimension of x and y, respectively. We will investigate the size of workgroup in the future.

The main operations of both 4×4 and 8×8 kernels are shown in Listing 1. First, the index of threads and workgroups are calculated to locate appropriate portion of the input. Second, input coefficients are loaded in rows per thread from the global memory into the private memory and row transform (1D-IDCT) is performed. However, when proceeding to the column transform, the coefficients in the same column are spread across different threads. Therefore, a store to local memory is required first for sharing the data. We transpose the coefficients by storing them in column order. Then, a synchronization is applied to ensure transposed data is written to the local memory. Next, the transposed input is loaded into private memory and column transform is performed by each thread. Finally, each thread stores its results back to global memory in their original positions.

Figure 3 exemplifies the data flow across different memory layers for the 4×4 IDCT in the GPU. Coefficients in the same rows are labeled with the same

color. Numbers within each box indicates their original positions in global memory. Arrows between different memory layers represent the threads within the x dimension of the workgroup. The 8×8 IDCT works in a similar fashion, but has a different implementation of the transform (1D-IDCT) and has a size of $N=8$ instead of 4. We will refer to this version of kernels as “compacted”.

```

kernel void Idct_NxN(global short *InOut) {
    //workgroup dimension: 3*32 (Dimy*Dimx)
    local short Shared[3*32*N];
    short Private[N];
    int Tx = get_local_id(0), Ty = get_local_id(1);
    int Dimx = get_local_size(0);
    int Dimy = get_local_size(1);
    int TyOffset = Ty*Dimx*N;
    int TxOffset = Tx*N;
    int WGx = get_group_id(0), WGy = get_group_id(1);
    int WGsInWidth = get_num_groups(0);
    int WGIdx = WGy*WGsInWidth+WGx;
    int WGOffset = WGIdx*Dimy*Dimx*N;

    global short *TSrc = &InOut[WGOffset+TyOffset+TxOffset];
    Private[0:1:N] = TSrc[0:1:N];
    1D_IDCT(Private);
    //write in column order
    local short *TShared = &Shared[TyOffset+Tx];
    TShared[0:Dimx:N*Dimx] = Private[0:1:N];
    barrier(CLK_LOCAL_MEM_FENCE); //synchronization

    TShared = &Shared[TyOffset+Tx*N];
    Private[0:1:N] = TShared[0:1:N];
    1D_IDCT(Private);

    int DstBlock = Tx%(N*N);
    int DstCol = Tx/(Dimx/N);
    int Dst = DstBlock*N*N+DstCol;
    global short *TDst = &InOut[WGOffset+TyOffset+Dst];
    TDst[0:N:N*N] = Private[0:1:N];
}

```

Listing 1. Pseudo-code of IDCT kernel

2.3 Further Optimizations for Compacted Kernel

Several optimizations are applied on top of the compacted kernel. In Nvidia GPU, instructions are scheduled in groups of 32 threads, termed as warps [2]. Therefore, the synchronization can be removed, as the warp size is greater than 4 or 8. Second, we increase the size of workgroup to 192. This leads to more warps feeding the compute units, improving the utilization of GPU. Third, we increase the data granularity processed per thread by 4 and 2 times for 4×4 and 8×8 kernel, respectively. By doing this, the index calculation overhead is reduced, leading to less instructions executed overall. However, the granularity can not be increased too much, as the size of input transferred to GPU is rounded up to multiples of unit data mapped to one workgroup. Increased data granularity will introduce more unnecessary computation. Finally, in the compacted kernel, the results are not written back coalescingly to the global memory [2], each thread

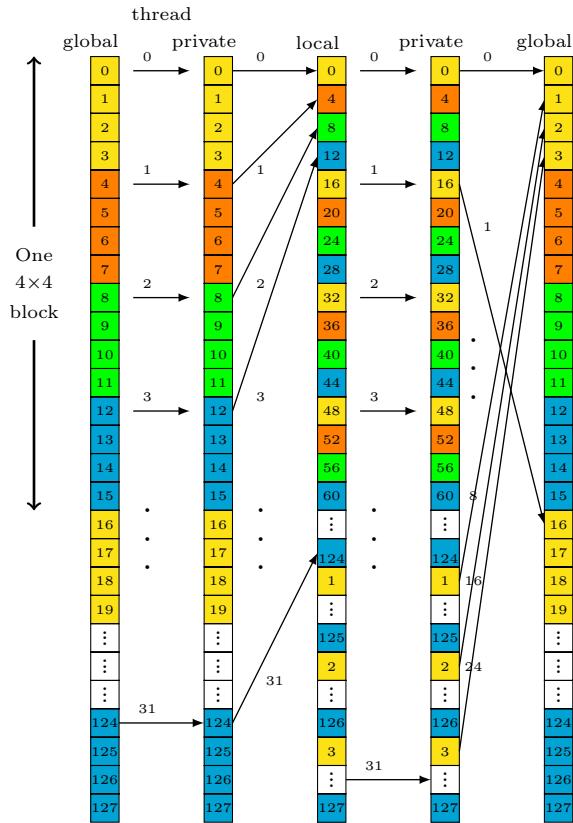


Fig. 3. Data flow for 4×4 IDCT in GPU

accesses addresses with a stride of 16 elements, as shown in Figure 3. To solve this problem, we rearrange the result first in local memory, with a padded stride to avoid the bank conflict in local memory, and then write them back to global memory in a coalesced way. This reduces the number of write requests to global memory.

3 Experimental Results and Discussion

We carry out our experiments on several hardware platforms consisting of different NVIDIA GPUs based on the Fermi Architecture [10]. Performance comparison is made against a highly optimized single-threaded CPU version executed on a Intel Sandybridge processor with SSE SIMD instructions, evaluated by the average time of five executions. Table 1 lists our hardware and software configuration. All results are obtained from GT430 except subsection 3.3.

Table 1. Experimental setup

System		GPU	
CPU	i5-2500K	GPU Architecture	Fermi
Frequency	3.3GHz	Compute capability	2.1
ISA	X86-64	GT430	Bandwidth:25.6GB/s
Operating system	Ubuntu 11.10	Shader cores:96	Frequency:1.4GHz
Linux kernel	3.0.0-13-generic	GTS450	Bandwidth:28.8GB/s
H.264 encoder	x264 0.115.1937	Shader cores:192	Frequency:1.56GHz
Compiler	GCC-4.4.6	GTX560Ti	Bandwidth:131.33GB/s
Optimization	-O2	Shader cores:384	Frequency:1.64GHz
Nvidia driver	280.13	OpenCL verison	1.1
CUDA toolkit	4.0	OpenCL build options	-cl-mad-enable

In order to cover videos with different characteristics we selected four (1920×1080 p) videos (*blue_sky*, *park_joy*, *pedestrian_area* and *riverbed*) and two (3840×2160 p) videos(*crowd_run* and *park_joy*).

The performance of the optimized IDCT depends heavily on the non-zero block ratio, and this, in turn depends on the encoding options. To cover different application scenarios we encoded all videos using three different encoding modes. The first two modes use a constant quality encoding mode (referred to as CRF) with different quality settings, CRF22 and CRF37. CRF22 generates higher quality videos at the cost of higher bitrate, and it is representative of high quality video applications. CRF37 increases the compression level at the cost of quality losses, and is representative of low to medium quality Internet video. The third mode is based on constant bitrate encoding with Intra-only prediction in which no samples from other frames will be used. The encoding options are listed in Table 2.

Table 2. Encoding parameters

Option	Value	Brief Description
Rate control	CRF-22, CRF-37	Constant quality encoding
	Intra-100	Constant bitrate at 100mbps for 1080p
	Intra-400	Constant bitrate at 400mbps for 2160p
Reference pictures	16	Number of reference pictures
Motion estimation	UMH	Uneven multi-hexagon
Motion search range	24	Max range of the motion search in pixels
Macroblock Partition	all	All macroblock partitions allowed

3.1 Effect of Compaction and Separation

Table 3 shows the effect of the compaction and separation optimization for 1080p and 2160p sequences with different encoding modes. Both the baseline and the compacted solutions are divided into three parts: kernel execution (Kernel), Host to Device (H2D) transfer, and Device to Host (D2H) transfer.

The kernel execution time is greatly reduced, proportional to the zero block ratio of different encoding modes. Along with the reduction of the kernel time, transfer time between CPU and GPU are reduced as well. In fact, they contribute more to overall speedup.

Table 3. Execution time (in ms) for baseline and compacted kernels

Res	Mode	Baseline			Compacted			Speedup		
		Kernel	H2D	D2H	Kernel	H2D	D2H	Kernel	H2D	D2H
1080p	CRF37	1.84	2.05	1.91	0.14	0.12	0.12	12.82	16.99	15.75
	CRF22	2.15	2.05	1.91	0.42	0.38	0.37	5.17	5.38	5.13
	Intra	2.59	2.07	1.91	0.84	0.79	0.78	3.08	2.63	2.46
2160p	CRF37	7.22	8.14	7.60	0.28	0.25	0.24	25.53	32.69	31.07
	CRF22	7.99	8.18	7.57	1.21	1.19	1.11	6.62	6.88	6.83
	Intra	10.85	8.23	7.58	3.84	3.94	3.57	2.82	2.09	2.12

3.2 Effect of Further Optimizations

Table 4 shows the effect of each of the optimization in section 2.3, in speedups over the previous one starting from the compacted kernel. Accumulated speedups of $1.51\times$ and $1.08\times$ are gained for 4×4 and 8×8 kernels, respectively. For synchronization removal, $1.05\times$ speedup is gained for 4×4 and $1.01\times$ for 8×8 kernel. The difference in the speedups are caused by the different computation granularities of the two kernels. For *workgroup* enlargement, $1.14\times$ is gained for 4×4 , compared to $1.01\times$ for 8×8 kernel. More instructions in 8×8 kernel can be executed in parallel, making it running efficiently on compute units even with less number of active warps. By enlarging the data *granularity*, relatively less instructions are reduced for the 8×8 kernel, compared to 4×4 kernel. Thus less speedup is gained. Finally, *coalescing* optimization contributes speedups of $1.02\times$ for 8×8 kernel and $1.07\times$ for 4×4 .

3.3 Performance of Optimized IDCT Kernel

The GPU IDCT kernels with all the optimizations enabled are compared against two CPU implementations: scalar and SIMD. For the SIMD implementation MMX and SSE are used to accelerate the 4×4 and 8×8 IDCT respectively. The performance of the SIMD and GPU kernel are presented in speedups over the scalar code. For scalability analysis purposes, the IDCT kernels are executed on three GPUs with numbers of shader cores 96, 192, and 384, respectively. Figure 4 shows the performance of 4×4 and 8×8 kernels as expected according to their computational capability. Maximum speedups over $25\times$ are observed for both kernels in GTX560Ti in Intra mode, because of its high bandwidth and large number of shader cores. The 4×4 kernel running on the GTS450 achieves a limited speedup compared to the speedup gained on the GT430. The low increase in memory bandwidth of the GTS450 over the GT430 limits the scalability for 4×4 kernel because of its high ratio of global memory access. GPU implementation

is faster than SIMD in all cases except the 8×8 kernel for CRF37 mode in the GT430. This results from the small data of the input, which also explains the low speedups gained in this mode. By contrast, maximum speedup is achieved in Intra mode, as kernel launch overhead is relatively small when computing a large amount of input data.

Table 4. Speedups of further optimizations

Kernel	Sync	Workgroup	Granularity	Coalescing	Total
4x4	1.05	1.14	1.19	1.07	1.51
8x8	1.01	1.01	1.04	1.02	1.08

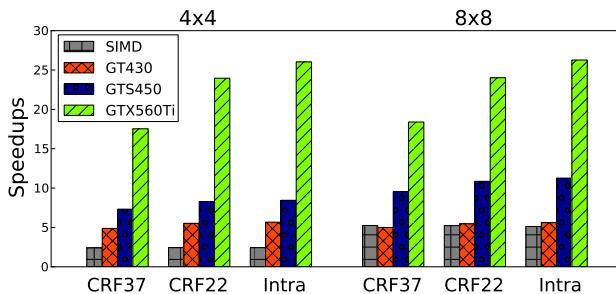


Fig. 4. Optimized kernel speedups over scalar code

3.4 Performance of Complete IDCT

The complete IDCT execution time, consisting of kernel execution time as well as memory transfer times between CPU and GPU, is presented in Table 3. Since modern GPUs are capable of concurrent memory copy and computation, data transfers can be overlapped with kernel computation. We perform overlapped execution between the 4×4 and 8×8 kernels. Speedups gained over scalar code for sequences encoded in different modes for 1080p and 2160p are presented in Figure 5. Overlapped execution gains speedups ranging from $1.2 \times$ to $1.3 \times$

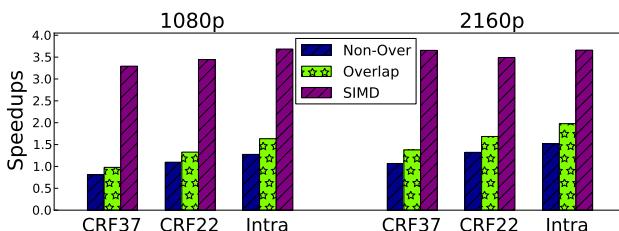


Fig. 5. Complete IDCT performance compared to scalar code

over non-overlapped execution. However, compared to SIMD code, the complete IDCT performance is slower because memory transfers still are the bottleneck for the overall performance.

4 Conclusions

In this paper we have exploited the fact that the input of the IDCT in H.264 contains a large number of zero coefficients and propose input compaction and separation to improve the GPU computation. Furthermore, additional optimizations are applied such as enlargement of data granularity and coalesced write back. The optimized GPU kernel shows a significant speedup compared to SIMD execution on the CPU, but the performance of complete IDCT is slower than the CPU SIMD version because of CPU-GPU memory transfer overheads. Our results do suggest, however, that kernels like H.264 IDCT could benefit from architectures with integrated CPUs and GPUs in which the cost of memory transfers is low.

References

1. Wiegand, T., Sullivan, G.J., Bjontegaard, G., Luthra, A.: Overview of the H.264/AVC Video Coding Standard. *IEEE Trans. on Circuits and Sys. for Video Technol.* 13(7), 560–576 (2003)
2. NVIDIA, NVIDIA CUDA C Programming Guide 4.2, http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
3. Khronos OpenCL Working Group, The OpenCL Specification 1.1, <http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf>
4. Malvar, H.S., Hallapuro, A., Karczewicz, M., Kerofsky, L.: Low-Complexity Transform and Quantization in H.264/AVC. *IEEE Trans. on Circuits and Sys. for Video Technol.* 13(7), 598–603 (2003)
5. Sullivan, G.J., Topiwala, P., Luthra, A.: The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In: SPIE Conf. on Applications of Digital Image Processing XXVII, pp. 454–474 (2004)
6. Chen, W.H., Smith, C., Fralick, S.: A Fast Computational Algorithm for the Discrete Cosine Transform. *IEEE Transactions on Communications* 25(9), 1004–1009 (1977)
7. Fang, B., Shen, G., Li, S., Chen, H.: Techniques for Efficient DCT/IDCT Implementation on Generic GPU. In: Proc. of the IEEE Int. Symp. on Circuits and Sys. (May 2005)
8. Obukhov, A., Kharlamov, A.: Discrete Cosine Transform for 8x8 Blocks with CUDA (October 2008), http://www.nvidia.com/content/cudazone/cuda_sdk/Image_Video_Processing_and_Data_Compression.html
9. FFmpeg, A H.264/AVC decoder, <http://ffmpeg.org/>
10. Wittenbrink, C.M., Kilgariff, E., Prabhu, A.: Fermi GF100 GPU Architecture. *IEEE Micro* 31, 50–59 (2011)

Multi-level Parallelization of Advanced Video Coding on Hybrid CPU+GPU Platforms

Svetislav Momcilovic, Nuno Roma, and Leonel Sousa

INESC-ID / IST-TU Lisbon,
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal

Abstract. A dynamic model for parallel H.264/AVC video encoding on hybrid GPU+CPU systems is proposed. The entire inter-prediction loop of the encoder is parallelized on both the CPU and the GPU, and a computationally efficient model is proposed to dynamically distribute the computational load among these processing devices on hybrid platforms. The presented model includes both dependency aware task scheduling and load balancing algorithms. According to the obtained experimental results, the proposed dynamic load balancing model is able to push forward the computational capabilities of these hybrid parallel platforms, achieving a speedup of up to 2 when compared with other equivalent state-of-the-art solutions. With the presented implementation, it was possible to encode 25 frames per second for HD 1920×1080 resolution, even when exhaustive motion estimation is considered.

1 Introduction

The increasing demand for high quality video communication, as well as the tremendous growth of video contents on Internet and local storages, stimulated the development of highly efficient compression methods. When compared to previous standards, H.264/AVC achieves compression gains of about 50%, keeping the same quality of the reconstructed video [1]. However, such efficiency comes at the cost of a dramatic increase of the computational demand, making real-time encoding hard to be achieved on single-core Central Processor Units (CPUs).

On the other hand, the latest generations of commodity computers, often equipped with both multi-core CPUs and many-core Graphic Processor Units (GPUs), offer high computing performances to execute a broad set of signal processing algorithms. However, even though these devices are able to run asynchronously, efficient parallelization models are needed in order to maximally exploit the computational power offered by these concurrently running devices. Such models must assure the inherent data dependencies in the parallelized algorithms, as well as a load balanced execution in the processing devices.

Recently, several proposals have been presented to implement parallel video encoders on GPUs [2–5]. However, most of them were only focused on a single encoding module. On the other hand, the task partitioning between the CPU and the GPU of these hybrid systems [6] has been referred to as the main challenge for efficient video coding on current commodity computers. The adopted

approaches [7, 8] usually offload most computationally intensive parts to the GPU, keeping the rest of the modules to be executed in the CPU. However, to the best of the authors' knowledge, there is not yet any proposal that effectively considers a hybrid co-design parallelization approach, where the CPU and the GPU are simultaneously used to implement the whole encoder structure.

By taking this idea in mind, an entirely new parallelization method is proposed in this paper. Such method is based on a dynamic performance prediction for parallel implementation of the entire inter-loop of the encoder, which simultaneously and dynamically exploits the CPU and the GPU computational power. In order to optimize such a hybrid platform, a dependency aware strategy for dynamic task and data distribution is proposed. The presented method relies on a realistic performance model that is built at run-time and improved at each iteration of the algorithm, in order to capture the real system behavior. For such purpose, it exploits several parallelization levels currently available in such a system, ranging from the fine-grained thread-level parallelism on the GPU, to both thread and vector-level parallelization on the CPU side.

2 Dependencies and Profiling Analysis of the H.264/AVC

According to the H.264/AVC standard [9], each frame is divided in multiple Macroblocks (MBs), which are encoded using either an intra- or an inter-prediction mode (see Fig. 1). In the most computationally demanding and frequently applied inter-prediction modes, the best-matching predictor of each MB is searched within already encoded Reference Frames (RFs). This process, denoted as Motion Estimation (ME), considers a further division of each 16×16 pixels MB into sub-blocks, as small as 4×4 pixels. The search procedure is then further refined by interpolating the RFs with half-pixel and quarter-pixel precision. Then, an integer transform is applied to the residual signal, which is subsequently quantized and entropy encoded, before it is sent alongside with the motion vectors (MVs) to the decoder. The decoding process, composed of the dequantization, inverse integer transform and motion compensation, is also implemented in the feedback

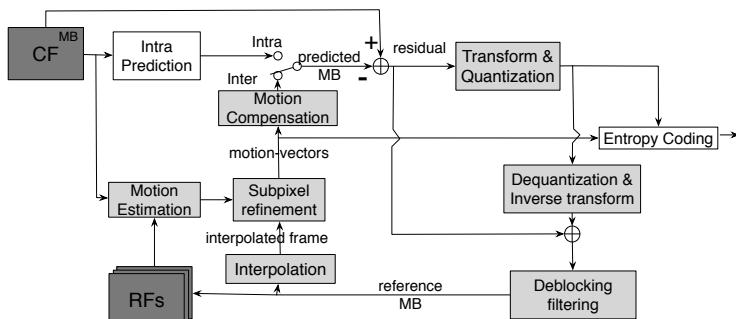


Fig. 1. Block diagram of the H.264/AVC encoder: inter-loop

loop of the encoder in order to locally reconstruct the RFs. A deblocking filter is finally applied to remove blocking artifacts from the reconstructed frame.

Several classes of data dependencies can be identified in this encoding process. *Inter-frame dependencies* exist between different frames in the video sequence. Such dependencies mainly arise from the ME procedure between the current and previously encoded RFs and limit the processing of successive video frames, by forcing them to be sequentially processed. *Intra-frame dependencies* exist between the processing of different regions of the same frame. They mainly exist in the intra-prediction encoding of the MBs, MVs prediction or in the deblocking filtering, when MB edges are filtered using the pixels of neighboring MBs. Functional *dependencies between the H.264/AVC modules* exist when the output data of one module represent the input data of another. For example, the MVs resulting from the ME define the initial search point for the Sub-pixel Motion Estimation Refinement (SME). Similarly, the sub-pixel values, obtained after the interpolation procedure, are the inputs of the SME. On the other hand, the interpolation and the motion estimation do not need to wait for each other, since both of them use the current frame and/or RFs.

From this dependencies analysis, it can be observed that parallel processing can only be considered within the scope of a single frame, since inter-prediction can not start before the list of RFs is updated. Moreover, due to the intra-frame dependencies in the deblocking filter, this module can not be concurrently applied on two different regions of each slice. Hence, the conjunction of all these observations exclude the possibility of dividing each slice in several independent parts to be simultaneously processed in a pipeline fashion. Finally, considering all the functional dependencies between the H.264/AVC modules, it can be concluded that only the interpolation and the ME can be processed in parallel, while the rest of the modules have to be sequentially processed. To simplify the presentation, and without any loss of generality, it will be assumed a single-slice frame configuration for the rest of this paper.

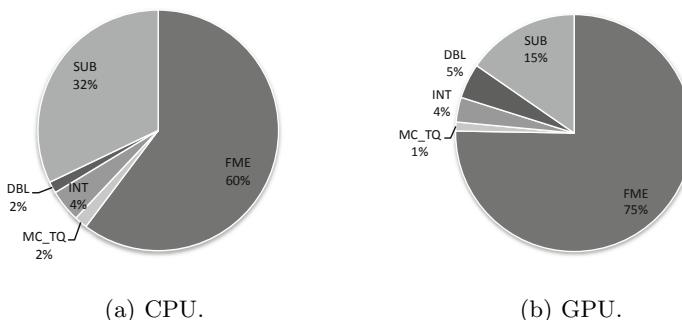


Fig. 2. Breakdown of the H.264/AVC inter-prediction-loop processing time (FME: full-pixel ME; SUB: sub-pixel ME; INT: interpolation; DBL: deblocking filter; MC_TQ: direct transform, quantization and inverse transform)

Fig. 2 represents a breakdown of the H.264/AVC processing time for both CPU and GPU implementations, regarding the several encoding modules. These profiling results were obtained for highly optimized implementations on an Intel Core i7 CPU and on an NVIDIA GeForce GTX 580 GPU. As it can be seen, ME is the dominant module, with more than 60% and 75% of the total processing time on CPU and GPU, respectively. Similarly, the SME also occupy a significant amount of the processing time. Finally, it is also observed that the conjunction of motion compensation, integer transform, quantization, dequantization and inverse transform, represented as MC-TQ, only represents about 1%-2% of the total processing time.

3 Performance Model and Task Distribution for Parallel Video Coding on a Hybrid CPU+GPU System

The heterogeneous structure of the H.264/AVC encoder includes modules with very different characteristics regarding the data dependencies and parallelization potential. In this section, it is analyzed the possibility of minimizing the encoding time by efficiently distributing the several tasks on the CPU and on the GPU.

3.1 Load Distribution and Scheduling Strategy

As a consequence of the profiling analysis presented in section 2, the most computationally demanding modules (ME and SME) are distributed among the CPU and GPU. These devices simultaneously execute these operations on different parts of the frame, where the frame division is considered to be at the level of rows/columns of MBs. This distribution is performed in a rather dynamic way, according to the performance level for each device that was evaluated in the previous encoded frame. Since the interpolation can be simultaneously executed with ME, this module is also considered when distributing the ME operation. A module-level scheduling is applied to the rest of the modules, in such a way that the overall processing time is minimized. Since all the other modules that can be concurrently processed (MC-TQ) only take 1% of the total time, their distribution among the CPU and GPU would not offer any significant advantage. Nevertheless, their execution is still evaluated in both processing devices (by using a predefined set of test frames), in order to predict further performance gains.

Algorithm 1 presents the implementation of the proposed method. The most complex steps will be explained in the following subsections. Before the encoding starts, the ME and SME modules are set to be performed on both devices on different halves of the frame, in order to perform a preliminary performance evaluation (lines 1 and 2 initializes the number of MBs assigned to each device). For the same reason, the rest of the modules are also assigned on both devices (line 3) and subsequently implemented in parallel (lines 5-13). Then, according to the measured times (line 14), it is decided which device will perform the interpolation operation (line 15). After that, the number of MB-rows to be sent

Algorithm 1. Scheduling/load balancing algorithm

```

1:  $n_{me\_cpu} = n_{me\_gpu} = \#MB_{rows}/2$ 
2:  $n_{sub\_cpu} = n_{sub\_gpu} = \#MB_{rows}/2$ 
3: Assign the rest of the modules to both devices
4: for  $frame\_nr=0$  to  $nr\_of\_frames$  do
5:   in parallel on CPU and GPU do
6:     Load  $n_{me\_cpu/gpu}$  rows
7:     Perform ME on  $n_{me\_cpu/gpu}$  rows
8:     Perform interpolation on assigned device(s)
9:     Exchange results
10:    Perform SUB on  $n_{sub\_cpu/gpu}$  rows
11:    Exchange results
12:    Perform the rest of the modules on assigned device(s)
13:  end in parallel
14:  Update times
15:  Decide device for interpolation
16:  Update  $n_{me\_cpu}$  and  $n_{me\_gpu}$ 
17:  Update  $n_{sub\_cpu}$  and  $n_{sub\_gpu}$ 
18:  if next frame is test-frame then
19:    Assign the remaining modules to both devices
20:  else if frame is the last test-frame then
21:    Perform the scheduling algorithm for remaining modules
22:  end if
23: end for

```

to each device is updated (see section 3.2) for both ME and SME (lines 16 and 17). Finally, whenever the following frame is marked as a test frame, the modules that will not be divided between the processing devices (all except the ME and SME) are assigned to both of them, in order to update their evaluation. Otherwise, they are distributed among the devices, in order to minimize the overall processing time. For this distribution, both the processing time and any eventual data transfer time are considered (see section 3.3).

3.2 Dynamic Load Balancing for Motion Estimation and Sub-pixel Motion Estimation Refinement

The distribution of the large computational load that is involved in the ME and SME modules among the CPU and GPU devices is performed at the level of MB-rows, by sending n_{gpu} rows to the GPU, and $n - n_{gpu}$ rows to the CPU, where n is a total number of MB-rows in each frame. By assuming that the attained performance (s), expressed as the number of processed MB-rows per second, does not depend on the considered distribution (i.e., $s_{gpu}, s_{cpu} = c^{te}$), the ME processing times on the two platforms can be expressed as:

$$t_{gpu_me} = \frac{n_{gpu}}{s_{gpu}}, \quad t_{cpu_me} = \frac{n - n_{gpu}}{s_{cpu}} \quad (1)$$

Hence, the main aim of the dynamic load balancing is to find the optimal distribution of MBs that will provide the most balanced execution time on the two processing devices, as stated in eq. 2. In this equation, t_{cpu_me} and t_{gpu_me} represent the processing time of the ME module on the CPU and the GPU, respectively. Conversely, t_{gpu0} and t_{cpu0} represent the execution time of the remaining processing modules that are supposed to be implemented on the CPU and GPU, together with the ME. As an example, such set of modules can include the interpolation operation, which does not have any data dependency with the ME and is required to be executed on one of these devices. Hence, the assignment of the interpolation module is done according to the ratio of performances on the two devices (i.e. speedup). In particular, if the interpolation is predicted to have a larger speedup than the ME module when sending it from the CPU to the GPU, it is assigned a greater offloading priority. Otherwise, it remains in the CPU, while the ME is simultaneously performed on the GPU. As soon as the interpolation is finished, the ME starts to be simultaneously executed on both devices.

$$t_{cpu_me} + t_{cpu0} \approx t_{gpu_me} + t_{gpu0}, \quad (2)$$

By combining eq. 1 with eq. 2, it is obtained:

$$\frac{n - n_{gpu}}{s_{cpu}} + t_{cpu0} \approx \frac{n_{gpu}}{s_{gpu}} + t_{gpu0}, \quad (3)$$

where:

$$n_{gpu} \approx \frac{n + s_{cpu}(t_{cpu0} - t_{gpu0})}{1 + \frac{s_{cpu}}{s_{gpu}}}. \quad (4)$$

However, the measured performance in any real system varies along the time, not only because of the changes in the conditions it operates on, but also because the inherent processing can be data-dependent. Therefore, if the number of MB-rows that is assigned to each device is computed by assuming the encoding time of a single frame, the obtained distribution will hardly be accurate along the time. As a consequence, the number of MB-rows that is submitted to the GPU should be updated in every iteration:

$$n_{gpu}^i \approx \frac{n - s_{cpu}^{i-1} \Delta t_0}{1 + \frac{s_{cpu}^{i-1}}{s_{gpu}^{i-1}}} , \quad n_{cpu}^i = n - n_{gpu}^i \quad (5)$$

where $\Delta t_0 = t_{cpu0} - t_{gpu0}$ is the signed sum of distribution-independent task portions on the CPU (positive sign) and on the GPU (negative sign). The measured performance values (s_{cpu}^{i-1} and s_{gpu}^{i-1}) are updated upon the encoding of each frame, according to the measured ME processing time on both devices. Hence, this iterative procedure starts with a predicted value (e.g., $n_{gpu}^0 = \#MB_{rows}/2$) and is updated until it converges to the ideal distribution, based on the measured performance on both processing devices.

The same strategy is applied in the case of the SME module. Since there is no other dominant H.264/AVC module that can be processed in parallel with

SME, the values of t_{cpu0} and t_{gpu0} will be set to zero, while the values of t_{cpu_sme} , n_{cpu_sme} , t_{gpu_sme} and n_{gpu_sme} will be updated along the time.

3.3 Scheduling of the Remaining Modules

As it was described in section 3.1, the least computationally intensive modules of the encoder are scheduled to be completely executed on one of the processing devices. The same happens with the deblocking filter, whose implementation can not be efficiently split by multiple devices. The proposed scheduling scheme is then applied, in order to minimize the overall processing time. For such purpose, all these modules are implemented and evaluated on both the CPU and GPU, and the measured processing times are then used as input parameters for the distribution algorithm, altogether with the data transfer times, required for any module transition between the devices.

The proposed distribution procedure is illustrated in Fig. 3. A data-flow diagram for all the encoding modules, considering both the CPU and GPU, is initially constructed. The transform and quantization tasks, as well as the dequantization and inverse transform, are presented together, due to the low computational requirements and simpler parallelization model. When the measured execution times are considered as a parameterization of each task, a weighted Directed Acyclic Graph (DAG) is obtained. The several nodes of such a graph (A, B ... H) are the decision points, where each task can be submitted to any of the two processing devices. The edges represent the individual task transitions, weighted by the respective computing and data transfer times. The shortest path between the starting and ending nodes of this graph represents the minimum encoding time. Dijkstra's algorithm [10] is typically used to find such a path, by

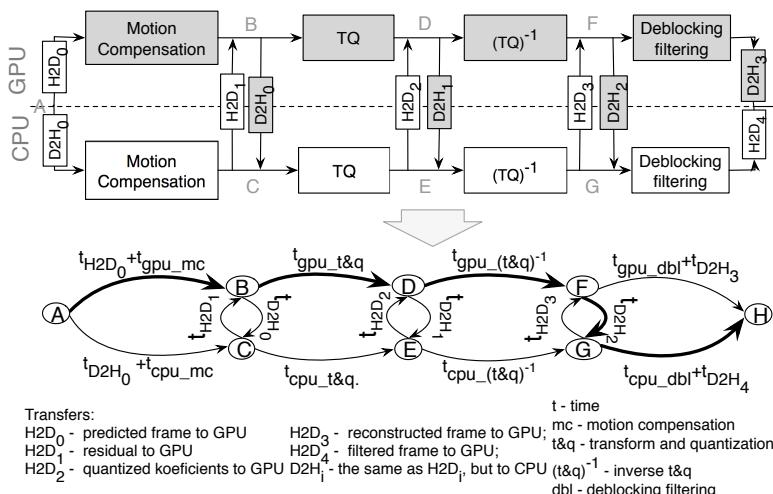


Fig. 3. Construction of weighted DAG from the data-flow diagram of H.264/AVC and a possible minimal path (represented in bold)

defining, for each encoding module of the inter-loop, the processing device on which it will be executed when encoding the subsequent frames. Due to the small number of nodes and edges, the application of this algorithm does not add a significant delay to the encoding procedure.

4 Experimental Results and Evaluation

The validation and evaluation of the proposed dynamic load distribution model was conducted with the H.264/AVC encoder implemented by the JM 17.2 reference software [11]. The considered test video sequences were *blue_sky*, *rush_hour* and *river_bed*, with a spatial resolution of 1920×1080 pixels. The ME module was parametrized with a search area of 32×32 pixels. The used evaluation platforms (presented in Table 1) adopted Linux operating system, CUDA 4.1 framework, *icc* 12.0 compiler and OpenMP 3.0 API to parallelize the video encoder. As it can be seen, *Platform 1* has a slightly faster GPU than *Platform 2*, and a significantly less powerful CPU.

Table 1. Hybrid (CPU+GPU) platforms adopted in the considered evaluation

	Platform 1		Platform 2	
	CPU	GPU	CPU	GPU
Model	Intel Core i7	GeForce 580GTX	Intel Core 2 Quad	GeForce 580GTX
Cores	4	512	4	512
Frequency	3GHz	1.54 GHz	2GHz	1.59GHz
Memory	4GB	1.5GB	4GB	1.5GB

The achieved encoding performance is presented in Fig. 4, for both hybrid platforms. The presented charts compare the resulting performance (in the time per frame (ms)) of five different scheduling strategies:

- CPU-only* - the whole encoder is implemented in the CPU;
 - GPU-only* - the whole encoder is implemented in the GPU;
 - Chen_original* - method proposed by Chen [7], where the ME, SME and interpolation modules are statically offloaded to the GPU (the rest are kept in the CPU);
 - Chen_optimized* - Chen's encoder [7], optimized with OpenMP and SSE4 vectorization techniques;
 - Proposed* - presented dynamic load distribution strategy.
- Contrasting to Chen's approach [7], which considers a static offloading to the

GPU of only the ME, SME and interpolation modules, the proposed distribution method combines an adaptive data-level partitioning (see eq. 5) and a dynamic selection of the device that offers the best performance for each of the processing modules of the video encoder (see Fig. 3).

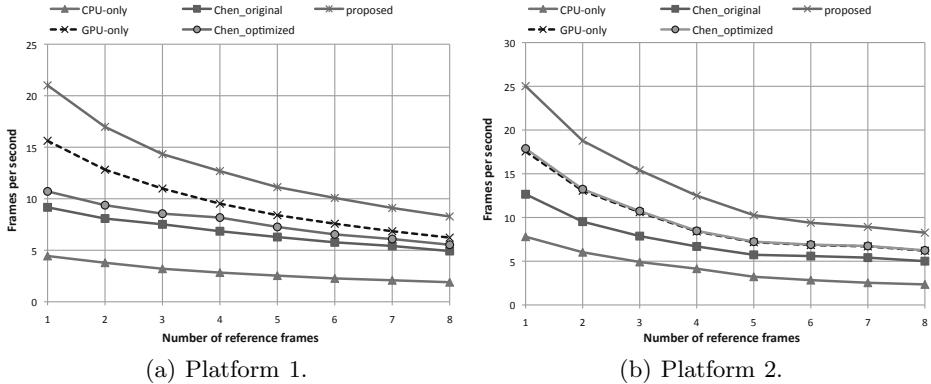


Fig. 4. Encoded frames per second (fps) for a varying number of RFs, using the 1920×1080 video format. Comparison of the proposed approach with CPU-only, GPU-only and the approach proposed by Chen [7].

The ME module of all these implementations (except the *CPU-only*) adopted the improved search algorithm proposed in [3]. Furthermore, OpenMP parallelization techniques to exploit the available number of CPU cores were extensively considered (except in *GPU-only* and *Chen_original*), as well as a broad set of CUDA optimizations to exploit, as much as possible, the GPU computational resources (except in *CPU-only*).

The performance regarding the frames per second considering different number of RFs are presented in Fig. 4 for two platforms specified in Table 1. As it can be seen, the *Proposed* method achieves speedup levels of up to 1.5 and 2, when compared with the *GPU-only* implementation and with the *Chen_optimized* strategy, respectively, and a speedup of up to 5 when compared with the *CPU-only* implementation. Due to the fact that *Platform 1* has a significantly slower CPU, the *Chen_optimized* strategy achieves a lower performance when compared to the *GPU-only* implementation, while in the case of *Platform 2* their performances are very similar. However, *GPU-only* requires a CUDA implementation of all the H.264/AVC modules of the inter-loop. The impact of the considered OpenMP and SSE4 vectorization [12] optimizations of the CPU code are emphasized when comparing the results obtained for the *Chen_original* and *Chen_optimized* approaches.

Contrasting with *Chen_optimized* and *GPU-only* approaches, the *Proposed* algorithm achieves a higher performance that is less dependent on the adopted hybrid platform. In particular, by simultaneously using the computational resources of the CPU and the GPU devices, in an adaptive and dynamic load balanced fashion, both processing devices participate in a much better distribution of the computational load, based on a constantly updated prediction of the offered performance by each device. Finally, it can be seen that a real time encoding with more than 20 fps is achieved on both platforms for a single RF.

5 Conclusions

A new dynamic load distribution model for hybrid CPU+GPU advanced video encoders was proposed. The distribution is carried out by exploiting both intra/inter task-level and data-level parallelism. The possibility of asynchronously processing on the CPU and GPU is also fully exploited to efficiently distribute the computational load of the most demanding H.264/AVC modules among these devices. A dynamic load balancing strategy is defined based on a performance model that uses prediction techniques from the previous processing results. Based on the proposed model and method, a speedup up to 2 for the total encoding time comparing state-of-the-art approaches was achieved, as well as the ability to encode more than 25 fps for a HD 1920×1080 resolution, considering all the sub-block prediction modes and an exhaustive ME algorithm.

Acknowledgment. This work was partially supported by national funds through Fundação para a Ciéncia e a Tecnologia (FCT), project PEst-OE/EEI/LA0021/2011.

References

1. Wiegand, T., Schwartz, H., Kossentini, F., Ulivan, G.S.: Rate-constrained coder control and comparison of video coding standards. *IEEE Transactions on Circuits and Systems for Video Technology* 13(7), 668–703 (2003)
2. Schwalb, M., Ewerth, R., Freisleben, B.: Fast motion estimation on graphics hardware for H.264 video encoding. *IEEE Transactions on Multimedia* 11(1), 1–10 (2009)
3. Momcilovic, S., Sousa, L.: Development and evaluation of scalable video motion estimators on GPU. In: *Workshop on Signal Processing Systems, SiPS* (October 2009)
4. Obukhov, A., Kharlamov, A.: Discrete cosine transform for 8x8 blocks with CUDA. Research report, NVIDIA, Santa Clara, CA (February 2008)
5. Pieters, B., et al.: Parallel deblocking filtering in MPEG-4 AVC/H.264 on massively parallel architectures. *IEEE Transactions on Circuits and Systems for Video Technology* 21(1), 96–100 (2011)
6. Cheung, N.M., Fan, X., Au, O.C., Kung, M.C.: Video coding on multicore graphics processors. *IEEE Signal Processing Magazine* 27(2), 79–89 (2010)
7. Chen, W.N., Hang, H.M.: H.264/AVC motion estimation implementation on Compute Unified Device Architecture (CUDA). In: *International Conference on Multimedia and Expo, ICME*, pp. 697–700 (April 2008)
8. Rodriguez-Sánchez, R.: Optimizing H.264/AVC interprediction on a GPU-based framework. *Concurrency and Computation: Practice & Experience* (2012)
9. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., Stockhammer, T., Wedi, T.: Video coding with H.264/AVC: tools, performance, and complexity. *IEEE Circuits and Systems Magazine* 4(1), 7–28 (2004)
10. Dijkstra, E.W.: A note on two problems in connexion with graphs. *Numerische Mathematik* 1(1), 269–271 (1959)
11. ITU-T: JVT Reference Software unofficial version 17.2 (2010),
<http://iphone.hhi.de/suehring/tm1/download>
12. Intel: SSE4 Programming Reference (2007),
<http://edc.intel.com/Link.aspx?id=1630>

Multi-GPU Implementation of the NICAM Atmospheric Model

Irina Demeshko¹, Naoya Maruyama², Hirofumi Tomita²,
and Satoshi Matsuoka¹

¹ Tokyo Institute of Technology, Tokyo, Japan

² RIKEN Advanced Institute for Computational Science, Kobe, Japan

Abstract. Climate simulation models are used for a variety of scientific problems and accuracy of the climate prognoses is mostly limited by the resolution of the models. Finer resolution results in more accurate prognoses but, at the same time, significantly increases computational complexity. This explains the increasing interest to the High Performance Computing (HPC), and GPU computations in particular, for the climate simulations. We present an efficient implementation of the Nonhydrostatic ICosahedral Atmospheric Model (NICAM) on the multi-GPU environment. We have obtained performance results for the number of GPUs up to 320. These results were compared with the parallel CPU version and demonstrate that our GPU implementation gives 3 times higher performance over parallel CPU version. We have also developed and validated the performance model for a full-GPU implementation of the NICAM. Results show 4.5x potential acceleration over parallel CPU version. We believe that our results are general, in that in similar applications we could achieve similar speedups, and have the ability to predict its degree over CPUs.

Keywords: GPU computations, CUDA Fortran, climate simulations, nonhydrostatic model.

1 Introduction

Climate change has significant impact on the Earth and human life as well as on the world's economic and geopolitical landscapes. The consequences of climate change such as higher temperatures, changing landscapes and different weather cataclysms potentially affect everyone. Thus, it is important to improve the understanding of the changing climate system and enable scientists to predict future climate behavior. Complex climate simulation modeling requires state-of-the-art HPC systems in order to get the "realistic" results. Particularly heterogeneous systems, which are based on using both conventional microprocessors and graphic processor units (GPUs), give us an extra performance for many kinds of scientific computations.

The main objective of this work is to increase performance of the existing climate simulation by using large-scale parallelism available on multi-GPU

environments. In this work we have developed and optimized a multi-GPU implementation of the ultra-high resolution atmospheric global circulation model NICAM [1] (see sections 3, 4). We have also built a performance model, which allows us to estimate potential performance of full-GPU implementation for any problem configuration (see section 5).

Our method is based on the localization of the most computationally intensive part of the climate model and modifying it to be computed on GPUs. One of the main challenges which such method faces is to reduce communication overheads and, at the same time, use large-scale parallelism of the hybrid system in the most efficient way. In order to achieve an optimal performance GPU kernels were analyzed and optimized. Experiments demonstrate that we have achieved significant acceleration of the initial parallel code. We have compared maximum performance for the initial MPI-parallel code with the one for our GPU implementation. The evaluation results show that the performance of the ported module is close to maximum efficiency and we can see more than 3x speedup in case of porting only one module to GPUs. Estimation results for a full-GPU implementation of shallow water NICAM simulation show potential 4.5x speedup over CPU MPI-parallel version.

Our contributions can be summarized as follows:

- We propose an effective multi-GPU implementation for a high resolution nonhydrostatic atmospheric model, specifically optimized for the large hybrid systems.
- We have evaluated our approach experimentally on more than hundred of nodes with 320 GPUs on a TSUBAME2.0 supercomputer. Evaluation results show performance's raise and significant acceleration of the original MPI-parallel code.
- We have developed performance model for a full-GPU NICAM implementation, which produces an estimation of potential performance for any problem configuration. It was shown that performance model prediction matches to the observed results for a large scale NICAM code.

2 Related Work

Increasing interest to the HPC among developer of climate simulation software can be explained by the extreme necessity to increase computational performance in purpose to improve the accuracy of simulations.

There are several weather/climate simulation models, which were recently accelerated with GPUs, such as: WRF[2], COSMO[3], NIM[4], HILRAM[5], ASUCA [6], GEOS-5[7] and GRAPES[8]. ASUCA, GEOS-5 and GRAPES are full-GPU approaches, WRF and COSMO(1) are based on porting only "Physics" module to the GPU, and COSMO(2), NIM and HILARM are poring to GPU only "Dynamics" module. Performance results are different for all listed simulation models and depends on numerical scheme as well as on the GPU-implementation strategy. Full-GPU implementation of ASUCA shows 26.3x speedup for a double precision and 80x speedup for a single precision cases[6]. By using GPU

co-processor technology GEOS-5 have demonstrated a potential speedup of 15-40 times over conventional processor cores[7]. Recent results of porting the GRAPES model to the GPU system show that acceleration was not as efficient as it was supposed and implementation algorithm needs significant optimizations. For the models, based on porting only one module (Physics or Dynamics) to the GPU we also observe quite sufficient acceleration, but the performance results are lower then for a full-GPU ones. It can be explained by higher communication overheads and smaller commutation intensity on the GPU. WRF GPU implementation shows 17x speedup over 1 CPU[2]; COSMO simulation on 1 GPU is 2x faster then on 6 CPUs; HILARM single GPU implementation shows 10x speedup over 1 CPU[5].

Our multi-GPU implementation of the NICAM is based on porting shallow water computations module (2-dimension "Dynamics" module) to GPUs and shows potential speedup of 4.5 times over CPU MPI-parallel implementation.

In our previous work [9] we have implemented the module, which performs the NICAM's main shallow water computations, on a single GPU. Due to the relatively small GPU memory, the results, we obtained, were limited only to small-scale problems (grid level up to 8). Multi-GPU implementation, described in this paper, solves the memory problem by distributing grid elements between nodes. We have modified our single GPU implementation into multi-GPU one. For this purpose we had to study the way to organize MPI-CUDA coordination efficiently.

3 The NICAM Model

NICAM is a Nonhydrostatic ICosahedral Atmospheric Model, which is designed to perform "cloud resolving simulations" by directly calculating deep convection and meso-scale circulations[1], [10]. Detailed description of the NICAM numerical scheme can be found in [1], [11], [12]. For the horizontal discretization NICAM model uses icosahedral grid system on the sphere. Grid dimension is defined by the grid division level n ($gl\ n$), and total number of grid point can be calculated by the formula: $Ng = 10(2^n)^2 + 2$

Initial NICAM code is based on FLAT-MPI parallel programming model. NICAM uses 2D decomposition, where initial grid is divided by regions. This regions are distributing between MPI processes. Total number of rectangles for the region level n can be calculated by the formula:

$$Nr = 10(4^n) \quad (1)$$

In this work we have investigated 2-dimensional (Shallow water) case of NICAM model [13], which plan to implement for the 3- dimensional case in a future work.

4 Implementation in CUDA Fortran

The CUDA architecture [14] enables hybrid computing, where both host (CPU) and device (GPU) can be used to accelerate different types of computations.

Current CUDA-enabled GPUs can contain from tens to hundreds of processor cores which are capable of running tens of thousands of threads concurrently. This gives us an opportunity to get significant acceleration of the code, but, at the same time, data transfers between host and device through the PCI bus tend to be the bottleneck in data movement. Therefore it is feasible to use GPUs only for computationally intensive code. The Portland Group (PGI) offers us an opportunity to explicitly program for GPUs using CUDA Fortran. NICAM is originally written in Fortran, thus the PGI CUDA Fortran syntax allows us to develop GPU kernels using a familiar coding environment.

Simplified flow-graph of the GPU NICAM implementation is presented on the Figure 1. It is shown that in the beginning of the computations we execute some initial modules like: MPI initialization, reading from the input file, checking the system, calculation of the grid geometry and some others. Main computations of the model are performing in the cycle by the time steps (nl).

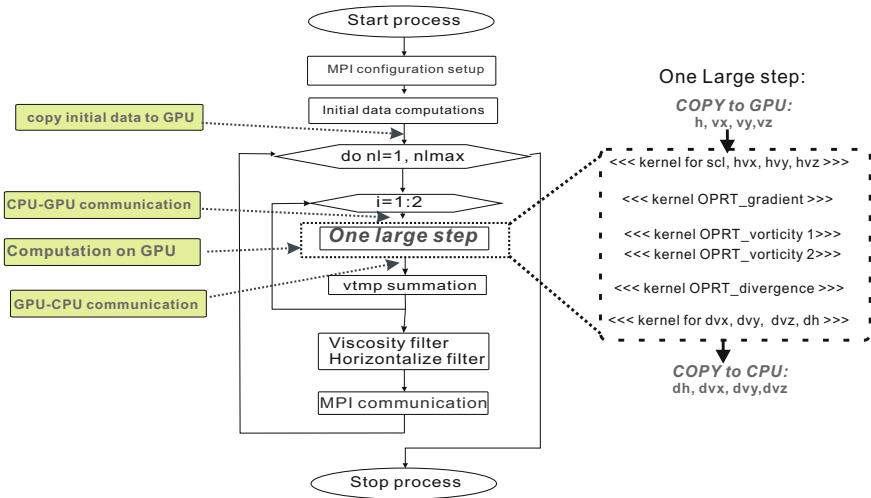


Fig. 1. GPU implementation scheme

In purpose to analyze runtime behavior of the given code we have used the Scalasca performance toolset [15]. According to the profiling results "One large step" is the most time consuming module of the code. It takes more than 50% of the whole time to compute this module. Therefore, in purpose to accelerate computations, we have decided to port this module to GPUs. We describe our GPU implementation algorithm below.

Before starting main cycle computations we send initial data arrays to GPUs (see Figure 1). This data are constant during entire computational process and we keep them in the GPU global memory until the end of the computations. The data, which are variable and necessary for the "One large step" module computations, are copying to GPUs in the beginning of the "One large step" module.

Then, after we finish computations on GPUs, we copy output data arrays to CPU. "One large step" module consist of 3 main subroutines: OPRT_gradient, OPRT_vorticity and OPRT_divergence. In purpose to reduce communication overheads we have also ported initial data and output data computation modules to GPUs and thus, keep all temporary data in the GPU global memory. Initial CPU code for this module was slightly modified in purpose to reduce the amount of memory to be allocated on GPU. We have created one kernel for OPRT_gradient and one for OPRT_divergence. For OPRT_vorticity module it was necessary to create 2 kernels, because we need to synchronize data within GPU in the middle of the vorticity module computations. Each modified module based on one nested loop or several loops calculation and each loop iteration computes 1 element of 2-dimensional array. Each thread of our GPU kernels calculates 1 element of the array.

CUDA programming model requires the programmer to organize parallel kernels into a grid blocks, which divided into thread blocks with at most 512 threads each. The NVIDIA GPU architecture executes threads of a block in SIMD (single instruction, multiple thread) groups of 32 called warps. NICAM model uses 2-dimensional grid, which size depends on the of grid level and region level sizes. We have used a block configuration of 256 threads, one thread per element. Our block size is a multiple of 32 which fits with the warp size and, therefore, allows us to achieve maximum efficiency. We have performed several tests to find the best way of organizing GPU-CPU communications and found that using pinned memory with PGI CUDA Fortran assignment gives as the best results and reduces communication time for about 3x time over a non-optimized version. By this means CPU-GPU communication bandwidth reached 5.6 GB/sec, which is close to the maximum. GPU kernels are located inside of each MPI process (see Figure 1) Therefore in the evaluation experiments we used the same number of MPI processes and GPUs.

It is shown in section 6 that CPU-GPU communications are quite significant and became a bottleneck. In purpose to increase performance we need to reduce communication time to computation time ratio. For this reason we plan to increase computational intensity on GPUs by porting the rest modules of the main Shallow Water computations to graphic processor units. In purpose to estimate potential performance for a full-GPU implementation of the NICAM we have built a performance model, described in the next section.

5 Performance Model

Simplified flow-graph for the full-GPU implementation of the NICAM is shown on the Figure 2 a). We assume that we compute "One large step", "Viscosity filter" and "Horizontalize filter" modules on GPUs. For this reason, we send h, vx, vy, vz arrays to GPU Global memory before starting GPU computations and copy output arrays dh, dvx, dvy, dvz to CPU after finishing "Horizontalize filter" computations. The size of arrays we copy to GPU and from GPU to CPU are the same as it was for a case of porting only "One Large step" module, and, therefore, the communication time (*COMM_time*) should be the same

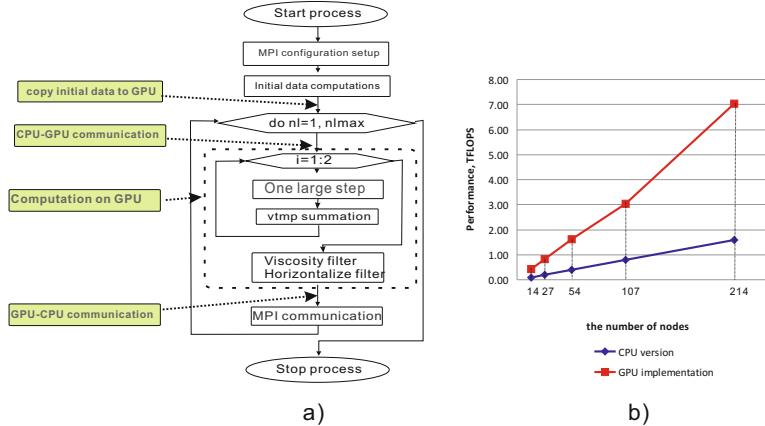


Fig. 2. Performance model a)flow-graph for a full-GPU NICAM implementation b) performance estimation results

The results we have got for the described multi-GPU implementation show that the "One large step" - is a memory bound module. Experimental results show, that its performance on GPU without communications tends to 14 TFLOPS (we will call it *OLSPerfomance* below). Also it was estimated that the average CPU-GPU communication bandwidth (*COMM_Bandwidth*) is about 5.6 GB/sec . The estimation of the potential performance of the full-GPU implementation is based on the number of floating point operations for entire shallow mode computation cycle (*FLOP_N*), obtained *OLSPerfomance* and *COMM_Bandwidth*, and was calculated by the formula (2):

$$\text{Performance} = \text{FLOP_N}/(\text{COMM_time} + \text{time_on_GPU}), \quad (2)$$

where $\text{COMM_time} = \text{transferred_memory}/\text{COMM_Bandwidth}$, $\text{transferred_memory} = 64 * ((2 + (\text{grid_level} - \text{region_level})^2)^2)$, Bytes, $\text{time_on_GPU} = \text{FLOP_N}/\text{OLSPerfomance}$

Size of the transferred memory depends on the grid dimension and, therefore, a function of the grid level and region level parameters. The formulas above estimate performance for any configuration of the problem size. Results are presented on the Figure 2 b). It is shown that the performance of the multi-GPU implementation potentially achieve 7 TFLOPS on 217 nodes, which is 4.5x higher than performance for the parallel CPU implementation. The performance model results were compared to the experimental ones (see section 6). We have observed that the model predictions match to the experimental results, which indicates that proposed model is valid and efficient.

6 Experimental Evaluation

In this work we have presented results of multi-GPU implementation of NICAM code on TSUBAME 2.0 supercomputer.

6.1 Environment

TSUBAME 2.0 consist of 1408 compute nodes of two Intel Xeon Westmere-EP 2.9 GHz CPUs and three NVIDIA M2050 GPUs with 52GB and 3GB of system and GPU memory, running SUSE Linux Enterprise Server 11 SP1. Each node has 2 sockets, 12 cores/node. We have used PGI CUDA Fortran compiler for the GPU code and PGI mpich2 compiler for the code on CPUs.

6.2 Results of the Multi-GPU Implementation

We have compared maximum performance, which is possible to get from each node, for the original MPI-parallel implementation of the code with the one for our multi-GPU implementation in our experiments. We have presented results for the grid levels 9, 10 and 11, which correspond to 2621440, 10485762 and 41943042 number of elements accordingly.

Due to the fact that we have 2 sockets per node, 6 cores per each socket and 3 GPUs per node, we have used 12 CPU cores per node for the CPU implementation and 3 CPU+GPU hybrid cores per node for the GPU version of the code. Initial NICAM code has a limitation in the number of MPI processes, which should be a division of the number of regions. The number of regions can be calculated by the formula (1) and only next numbers of processes are available: 1, 4, 5, 8, 10, 16, 20, 32, 40, 64, 80 and so on. According to the statements above we have used following configurations: we have compared 14 nodes with 160 CPU cores for the CPU implementation versus 40 CPU+GPU hybrid cores for the multi-GPU one; 27 nodes with 320 CPU cores versus 80 CPU+GPU hybrid cores; 54 nodes with 640 CPU cores versus 160 CPU+GPU hybrid cores, and 107 nodes with 1280 CPU cores versus 320 CPU+GPU hybrid cores. The results are presented on Figures 3, 4.

We present performance breakdown results for a GPU implementation in the Figure 3 a). It is shown that about 30-40% of the overall time of the GPU implementation is spent for CPU-GPU communications. This ratio is quite significant, which signify that communication overheads are the bottleneck for the described implementation. It was shown in the previous section that we achieve higher performance by increasing computational intensity on GPU. Therefore, we suppose that we will get much better results when we switch from the 2D case to the 3D one, because computation intensity on GPU for a 3D case is significantly higher than for a 2D implementation. We compare overall computation time for GPU implementation with the one for initial MPI-parallel version on the Figure 3 b) It is shown that our multi-GPU implementation is up to 3 times faster then the initial parallel CPU code. At the same time, we can observe that for smaller grid sizes (e.g. gl 09) acceleration is slightly lower and we have smaller execution time

difference between parallel CPU and multi-GPU implementations, which can be explained by lower computation intensity per node. Thus, the GPU implementation is more efficient for grid levels higher than 9, which is the lowest grid level required by the NICAM model to produce relatively "realistic" results.

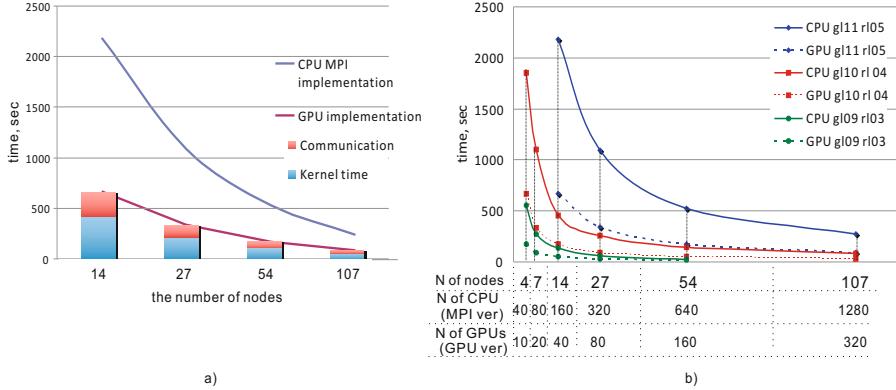


Fig. 3. Average running time for MPI-parallel CPU version versus time multi-GPU implementation of NICAM as a function of the number of nodes

Performance as a function of the number of nodes for a multi-GPU implementation and initial NICAM version is shown on the figure Figure 4a). The results are similar to the ones on the Figure 3 b) and performance for a GPU implementation is up to 3 times higher than for a MPI-parallel version.

Figure 4 b) compares performance we obtained for the proposed GPU implementation with the theoretical one, estimated by the performance model described at the section 5. According to the formulas from the section 5 and due

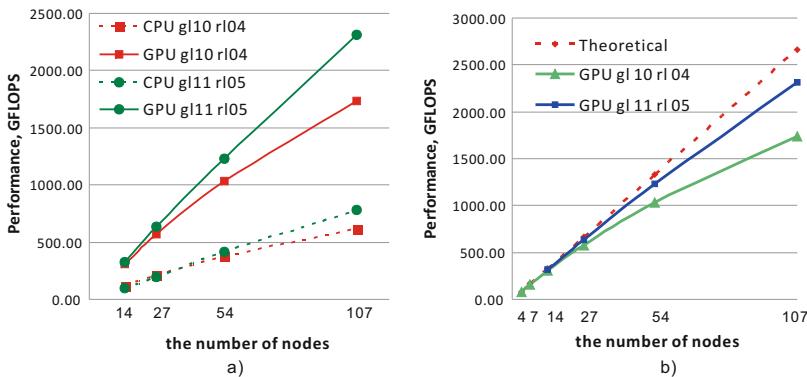


Fig. 4. Performance as a function of the number of nodes. a) Parallel MPI version versus multi-GPU implementation. b) Theoretical and experimental performances

to the fact, that increasing the grid level by 1 we 4x time increase computational intensity and transferred data size, theoretical performance is the same for any configuration of the problem. From the graph we can see, that the higher grid dimension the closer experimental performance to the theoretical one. This indicates that our theoretical model better fit to the real-size problem, due to we have higher computational intensity for a fine-grained tasks and less overheads. It can be observed that for the grid level 11 up to 54 nodes the experimental performance close to the theoretical one, which validates the proposed performance model. A little difference of the experimental performance for the 107 nodes with the theoretical one can be explained by the reduced computation intensity per node, which reduces efficiency of the GPU implementation. We have also performed some additional experiments to compare our performance model estimations with the experimental results. It was observed, that the model is well matching tho the experiments for such a large-scale model as NICAM. So, we believe that we can use the model for the purpose to estimate reliable performance results for a reasonable sized problem configuration.

7 Conclusion

Climate changes becoming a critical topic in everyday life and, therefore, an interest to climate simulation models is rapidly increasing. The climate models are tools that demand high performance computations to achieve reliable accuracy of the results. Heterogeneous systems are emerging as attractive computing platforms for HPC applications. Large-scale parallelism available on multi-node GPU environment gives us an opportunity to get better performance for many kinds of scientific computations.

In this paper we have presented results of the multi-GPU implementation of the Nonhydrostatic ICosahedral Atmospheric Model on a big heterogeneous system. We have ported the most time-consuming module of the initial code to GPUs by using PGI CUDA Fortran. We have demonstrated that our multi-GPU implementation gives 3 times higher performance comparing with the maximum performance of the MPI-parallel version. It was also shown, that performance of the proposed GPU implementation is almost optimal for "reasonable-sized" problems. We have developed a performance model for a full-GPU implementation. Estimation results show 7 TFLOPS potential performance on 217 nodes for a multi-GPU implementation, which is 4.5x times higher than the performance of the parallel CPU version. The model was validated and shows a good matching to the experimental results. The results above prove that proposed method is highly efficient and we can recommend it for many kinds of climate simulations.

We intend to apply described GPU implementation of NICAM shallow water code for a tree-dimensional NICAM climate simulation model, which would give us higher performance results due to increasing computational intensity per node. Also we plan to investigate performance of the NICAM code with OpenACC and compare it with the one, we obtained with PGI CUDA Fortran.

References

1. Satoh, M., Matsuno, T., Tomita, H., Miura, H., Nasuno, T., Iga, S.: Nonhydrostatic Icosahedral Atmospheric Model (NICAM) for global cloud resolving simulations. *Journal of Computational Physics, The Special Issue on Predicting Weather, Climate and Extreme events* 227, 3486–3514 (2007)
2. Michalakes, J., Vachharajani: GPU acceleration of numerical weather prediction. In: IPDPS, pp. 1–7. IEEE (2008)
3. COSMO-model (2012), <http://www.cosmo-model.org/>
4. Govett, M.W., Middlecoff, J., Henderson, T.: Running the NIM next-generation weather model on GPUs. In: Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud and Grid Computing, CCGrid, pp. 792–796 (2010)
5. Vu, V.T., Cats, G., Wolters, L.: GPU acceleration of the dynamics routine in the HIRLAM weather forecast model. In: HPCS 2010, pp. 31–38. IEEE (2010)
6. Shimokawabe, T., Aoki, T., Muroi, C., Ishida, J., Kawano, K., Endo, T., Nukada, A., Maruyama, N., Matsuoka, S.: An 80-Fold Speedup, 15.0 TFlops, Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code. In: The 2010 ACM/IEEE Conference on Supercomputing, SC 2010, pp. 1–11 (2010)
7. Putnam, W.: Graphics Processing Unit (GPU) Acceleration of the Goddard Earth Observing System Atmospheric Model. NASA Technical Report, Goddard Space Flight Center (2011), <http://ntrs.nasa.gov/search.jsp?R=20120009084/>
8. Wang, Z., Xu, X., Xiong, N., Yang, L.T., Zhao, W.: GPU Acceleration for GRAPES Meteorological Model. In: 2011 IEEE 13th International Conference on High Performance Computing and Communications, HPCC, pp. 365–372 (2011)
9. Demeshko, I., Matsuoka, S., Maruyama, N., Tomita, H.: Ultra-high resolution atmospheric global circulation model NICAM on graphics processing unit. In: Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications (2012)
10. Tomita, H., Satoh, M.: A new dynamical framework of nonhydrostatic global model using the icosahedral grid. *Fluid Dynamics Research* 34, 357–400 (2004)
11. Tomita, H., Tsugawa, M., Satoh, M., Goto, K.: Shallow water model on a modified icosahedral geodesic grid by using spring dynamics. *J. Comp. Phys.* 174, 579–613 (2001)
12. Satoh, M., Tomita, H., Miura, H., Iga, S., Nasuno, T.: Development of a global cloud resolving model – a multi-scale structure of tropical convections. *J. Earth Simulator* 3, 11–19 (2005)
13. Toro, E.F.: Shock-Capturing Methods for Free-Surface Shallow Flows, 309 pages. Wiley and Sons Ltd. (2001)
14. NVIDIA Corporation. NVIDIA CUDA Programming Guide, version 4.1 (2012), <http://www.nvidia.com/cuda>
15. The Scalasca performance toolset (2012), <http://www.scalasca.org/>

MPI vs. BitTorrent: Switching between Large-Message Broadcast Algorithms in the Presence of Bottleneck Links

Kiril Dichev and Alexey Lastovetsky

Heterogeneous Computing Laboratory
University College Dublin
Dublin, Ireland

`Kiril.Dichev@ucdconnect.ie, Alexey.Lastovetsky@ucd.ie`

Abstract. Collective communication in high-performance computing is traditionally implemented as a sequence of point-to-point communication operations. For example, in MPI a broadcast is often implemented using a linear or binomial tree algorithm. These algorithms are inherently unaware of any underlying network heterogeneity. Integrating topology awareness into the algorithms is the traditional way to address this heterogeneity, and it has been demonstrated to greatly optimize tree-based collectives. However, recent research in distributed computing shows that in highly heterogeneous networks an alternative class of collective algorithms - BitTorrent-based multicasts - has the potential to outperform topology-aware tree-based collective algorithms. In this work, we experimentally compare the performance of BitTorrent and tree-based large-message broadcast algorithms in a typical heterogeneous computational cluster. We address the following question: Can the dynamic data exchange in BitTorrent be faster than the static data distribution via trees even in the context of high-performance computing? We find that both classes of algorithms have a justification of use for different settings. While on single switch clusters linear tree algorithms are optimal, once multiple switches and a bottleneck link are introduced, BitTorrent broadcasts – which utilize the network in a more adaptive way – outperform the tree-based MPI implementations.

Keywords: BitTorrent, MPI, Broadcast, Bandwidth, Bottleneck Link.

1 Introduction and Related Work

The broadcast operation distributing data from a root process to all other processes is one of the fundamental collective operations in high-performance computing. Numerous research has been done to develop broadcast algorithms for various platforms over the last 20 years. This work focuses on Ethernet-switched networks – an overview of the most common MPI broadcasts and their performance for Ethernet networks can be found in [15]. The basic existing algorithms in the state-of-the-art MPI implementations are the flat tree (also called round-robin), linear tree (also called daisy chain or pipeline) and binomial tree algorithms.

For clusters connected through heterogeneous networks, very different strategies have emerged for large-message broadcasts. Tree-based algorithms can be optimized through topology awareness – however, computation of optimal communication trees is expensive to compute (NP-complete). Heuristics can alleviate this problem, and early work computing communication trees in polynomial time includes [9,2,1]. More recently, we have used models to also generate efficient communication trees [7]. As an alternative, simpler broadcast algorithms can be devised to reflect e.g. a two-layer hierarchy [12].

In the area of peer-to-peer computing, Burger [3,4] has recently demonstrated that receiver-initiated multicasts (of which BitTorrent is an example) can be very efficient. Experiments have shown that this class of algorithms has the potential to outperform even optimized MPI libraries on emulated heterogeneous networks.

Independently from this work, we have recently addressed a related problem in high-performance computing – why try to minimize the network utilization on clusters through complex tree-based algorithms instead of maximizing it? Our objective is the minimization of total communication time, and not the minimization of network utilization. We decided to experiment with the BitTorrent [5] protocol and use it in an unusual setting – for cluster communication. Our research in large message broadcast algorithms on hierarchical and heterogeneous networks shows an interesting and surprising way forward. We show that the protocol has a number of properties which contribute to maximizing the network utilization. We experimentally verify that BitTorrent broadcasts often outperform optimized tree-based broadcasts for different cluster settings. In addition, the original implementation of BitTorrent is oblivious of the network topology – which makes it trivial to deploy anywhere.

Table 1. Optimal large-message broadcast algorithms for the two extremes of homogeneous or heterogeneous networks according to recent research

Underlying network	Optimal broadcast algorithm
Very homogeneous	Linear tree algorithm (static)
Very heterogeneous	topology-aware pipelined trees (static) or receiver-initiated multicasts (dynamic)
Some level of heterogeneity	Unknown

Inspired by these developments – summarized in Table 1 – this work builds a bridge between the algorithms used in the high-performance computing domain and the algorithms used in the distributed computing domain for large message broadcasts. In particular, we examine if the described algorithms both have their justification when using settings which are neither fully homogeneous nor very heterogeneous. Some of the questions we answer in this work are:

1. *How sensitive is the performance of operations like linear tree and BitTorrent broadcasts to the heterogeneity of the underlying network? When does it make sense to switch between algorithms?*

This question is not addressed in any previous work to the best of our knowledge. The high-performance computing domain usually assumes a single switched cluster or a hierarchy without any underlying heterogeneity [13]. On the other hand, [3] assumes a high level of heterogeneity, with emulated cluster scenarios differing in the available link bandwidth around 10 times.

2. *Are the results of the related work on large-message broadcasts as shown in Tab. 1 in agreement with experiments on modern grid clusters with a moderate level of heterogeneity?*

This is an important experimental validation, since this study overlaps with existing research on broadcast operations performed in very different research domains and using different settings.

The paper is structured as follows - in section 2 we highlight the principle differences between the BitTorrent version and the main MPI versions of performing a broadcast operation. In section 3 we present our main experimental settings – with or without a bottleneck link – and our benchmark results. Section 4 concludes the paper.

2 Broadcast in BitTorrent and MPI

2.1 Complexity for Large Message Broadcasts

The classic three algorithms for broadcast operations in MPI – linear, binomial and flat tree algorithm – are described in detail in previous work [15]. A number of variations based on these algorithms exist – for example the scatter/allgather version used in MPICH2 for large messages, where the scatter operation uses a binomial tree, and allgather has a number of variants (e.g. ring implementation).

For large messages, fragmentation and pipelining always yield better performance and variations on that include pipelined linear tree algorithm, pipelined binomial tree algorithm and scatter/allgather-based broadcast. [13] describes pipelined algorithms and observes that on Ethernet switched clusters and for sufficiently large messages, the message size dominates the runtime (rather than the number of processes). Then the theoretical lower limit for a broadcast of a message is the transfer time of this message only between two nodes, since in a pipeline many nodes can overlap their message transfer to each other. The authors perform some simple analysis finding that a pipelined linear tree broadcast without contention and with good segment size comes close to the theoretical lower limit for large messages on single-switch clusters as well as on multi-switch clusters with fully homogeneous network. This is also experimentally confirmed.

In summary: When broadcasting very large messages (Megabytes) across 10s to 100s of processes, efficient algorithms like the linear tree algorithm in MPI have a time complexity of $O(M)$ with M being the message size. With the BitTorrent-based approach, we can only hope to reduce this complexity by a constant factor.

2.2 Algorithm Differences between MPI and BitTorrent Broadcasts

In the following, we focus on the significant differences between the data distribution in BitTorrent as compared to the MPI broadcast algorithms.

– Pipelining

Fragmentation and pipelining always lead to higher parallelism for large messages, and MPI and BitTorrent both use these techniques. The fragment size respectively is best determined at runtime. Our profiling shows that BitTorrent uses fragment size of 16 KB, and Open MPI uses fragment size of 128 KB.

– Parallel point-to-point transfers at a time

Another important parallelization aspect is how different point-to-point calls are parallelized, i.e. how many point-to-point transfers can be performed in parallel through the network. In MPI, the pipelined linear tree algorithm can theoretically provide a parallelism of $(p-1)$ point-to-point calls when the pipeline is full. In BitTorrent, a much denser (but not complete) communication graph can be used. The protocol can provide for $2 * p * c$ parallel point-to-point calls with c being the allowed active downloads or uploads at a time. For the original client we use, 4 parallel uploads are allowed (more downloads are allowed, but only the lower bound is relevant), i.e. $c = 4$. The constant 2 denotes that BitTorrent allows bidirectional message transfers. In theory, this means that BitTorrent could utilize the network better, particularly parts of the network in the presence of bottlenecks. In practice, 100 % parallelization of all point-to-point transfers can not be achieved, since a node can not physically send or receive data in parallel to all peers - the Ethernet adapter is one example of a serialization point. Furthermore, network saturation and even congestion are a possibility.

– Dynamic Parallel Access

Dynamic parallel access is a significant feature of BitTorrent not present in any of the MPI-based collective algorithms. In a BitTorrent protocol, a process can listen on a socket for a data chunk from several peers in parallel (see Fig. 1). Indeed, this is a very powerful feature which allows for dynamicity and adaptivity in communication as demonstrated for wide-area networks, e.g. in [14]. This is not possible in the sender-initiated MPI collectives where the schedule allows each process to receive data from only one peer.

– Tree vs. Complete Graph

All modern broadcast algorithms in MPI follow a communication tree. The flat tree and the linear tree algorithm can be seen as two opposite extremes of communication trees (with minimal and maximal depth), the binomial tree is another option. On the other hand, the BitTorrent algorithm builds a dense communication graph, in which every process can potentially exchange data with every other process (but only keeps a record of 35 peers in the used implementation).

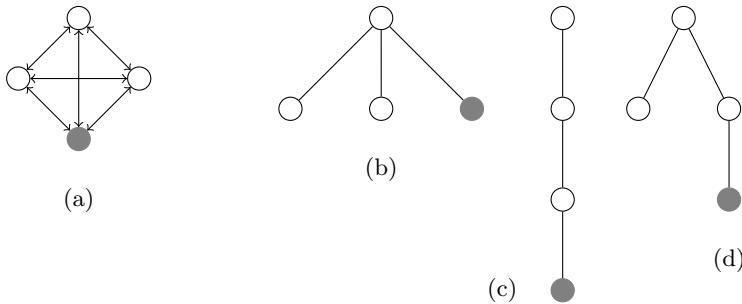


Fig. 1. Point-to-point communication in BitTorrent (a) and MPI broadcasts with flat tree (b), linear tree (c) or binomial tree (d). Only BitTorrent allows a random process (colored in gray) to receive fragments from any other process

3 Experimental Setup and Results

3.1 Bordeaux Site in Grid'5000 as Experimental Platform

We used the 3 clusters Bordereau, Borderline and Bordeplage in Bordeaux in Grid'5000 as experimental platform. Figure 2 shows the network between the clusters. Here, isolated point-to-point bandwidth between any two nodes across the clusters is 1 Gbps (limited by the network interface). However, when intense collective communication is used, the point-to-point throughput decreases significantly across the Dell ProConnect – Cisco connection , because only a single 1 Gbit link connects the two switches. In addition, we measure an increased latency along this link – easily explained with the traversal of more switches. This is the main potential bottleneck. To a lesser extent, the Nortel-HP link also can turn into a bottleneck link for collective communication.

The used setting has significantly less heterogeneous network properties than settings usually used in the distributed computing domain, but we do not consider this a disadvantage. On the contrary, this moderately heterogeneous setting is more typical for high-performance computing.

3.2 Modifications for Profiling BitTorrent and MPI Libraries

We use the original BitTorrent client written by Bram Cohen [5] for our experiments. It has Open Source License and is written in Python. While probably not as efficient as C/C++ compiled binaries, the Python scripts are convenient as proof of concept, and debugging and modifications are comparatively easy. We mostly used the source code itself as well as [10] as a reference for details on the algorithm. The software is available as a package in most modern Linux-based distributions.

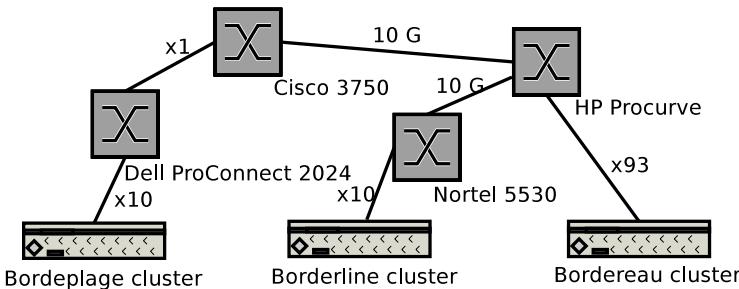


Fig. 2. Ethernet network on Bordeaux site. 10G denotes a single 10 Gigabit link; x1, x10 and x93 denotes Ethernet bonding of 1, 10 or 93 1 Gigabit connections. The Dell–Cisco link is a major bottleneck during intense collective communication between Bordeplage and Borderline or Bordeplage and Bordereau

The following modifications were introduced into the original BitTorrent client:

- File I/O was removed. Instead, dummy data strings are generated on-the-fly and transferred over the network.
- The wall clock time is taken at initiation of the class StorageWrapper and at download completion in the same class. The time difference is used as reference.

For discovering the runtime algorithm and fragment size for pipelining in Open MPI [8] we used PERUSE [11]. PERUSE is an event-driven tracing library for internal MPI events. We used it because the decision making process in MPI collectives is very complex and runtime checks with PERUSE are a reliable way to find which algorithm is used for particular process number and message size. For MPICH2, we use related work and the source code to find which algorithm is being used.

3.3 Timing Mechanism Used in BitTorrent and MPI

In this section, we give a detailed explanation of the timing methodology used consistently both in BitTorrent and MPI experiments. This is important since BitTorrent originally does not provide such timing, while MPI supports timing and logical operations on timings for communication calls.

The runtime setup for BitTorrent involves starting a BitTorrent tracker and then launching BitTorrent clients simultaneously identically to MPI program startup. The execution time of a BitTorrent program is then taken as the wall clock time between the start of a StorageWrapper instance and the moment download completion is registered. A BitTorrent client then has to be explicitly terminated since it has no concept of completing a collective communication. In MPI, a barrier call is made, and then the wall clock time is taken before and after the broadcast operation. Then, timing mechanism is as follows:

- In each run and for both types of broadcasts, 64 processes are run, and each of them provides a different wall clock time to finish. As a reference, we take the maximum time between all processes both for BitTorrent and MPI.
- We also perform a number of iterations for each run. As a reference, we take the average of all iterations – again, both for BitTorrent and MPI.

For each message size and each setting, we perform 5 iterations for MPI and BitTorrent.

3.4 Benchmarks of MPI and BitTorrent Broadcasts on a Homogeneous Setting

In the first setting, we test the performance of the presented broadcast algorithms without involving the main bottleneck link. We first use 64 nodes on the Ethernet cluster Bordereau (Fig. 3(a)). Then we use 55 nodes on Bordereau and 9 nodes on Borderline (Fig. 3(b)). We benchmark three broadcast versions - MPICH2, Open MPI and BitTorrent. The used message sizes are 1, 5, 10, 50, 100 and 239 MB. The results for both runs are similar. They demonstrate that for the message sizes 5 MB and 10 MB, MPICH2 marginally outperforms Open MPI, but Open MPI and the linear tree algorithm is most efficient for the rest of message sizes. However, the broadcasts with BitTorrent come very close to the Open MPI broadcasts and even outperform MPICH2 for large messages. This result is unexpected, since the initial assumption is that BitTorrent-based broadcasts are not suitable for homogeneous clusters – however, BitTorrent performs excellently for both settings. The difference to the linear tree broadcast is even minimal for the second run. We interpret this with the fact that this run involves the Nortel-HP link as well and this introduces an increase in network heterogeneity.

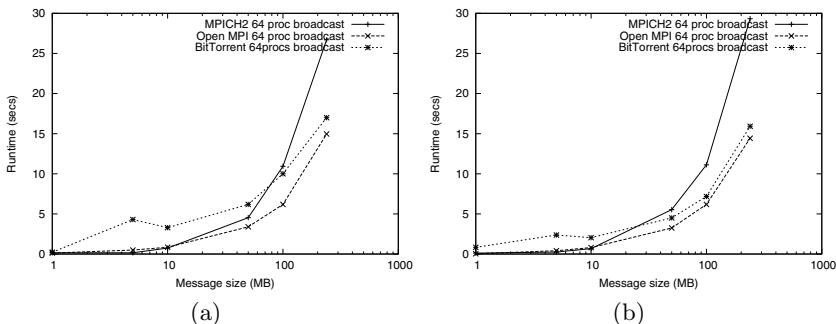


Fig. 3. 64 node broadcasts without the main bottleneck link: using single cluster Bordereau (a) or clusters Bordereau and Borderline (b). On this homogeneous setting, the linear tree algorithm performs best as expected for very large messages, but BitTorrent also performs well.

3.5 Benchmarks of MPI and BitTorrent Broadcasts on More Heterogeneous Settings

In the second setting, we involve the main bottleneck link (Fig. 2) in two different runs. First, we use 32 Bordeplage nodes and 32 Bordereau nodes (Fig. 4(a)). Then, we involve 32 Bordeplage nodes, 25 Bordereau nodes and 7 Borderline nodes (Fig. 4(b)). For the MPI runs, processes are started in the way they are listed. We consider this the most efficient process-to-node assignment for the linear tree algorithm. Inter-cluster communication is minimized, and intra-cluster communication has been proved to be efficient anyway. For MPICH2, there is no simple solution for providing an optimal file for the scatter/allgather algorithm, and we provide the same process-to-node mapping. We use the same broadcast implementations and message sizes as before. The benchmarks show that on both settings, for message sizes of 50 MB and larger BitTorrent (which is oblivious of the topology) outperforms both MPICH2 and Open MPI. A secondary result is that for the same large message range, the scatter-allgather algorithm used by MPICH2 outperforms the linear tree algorithm used by Open MPI.

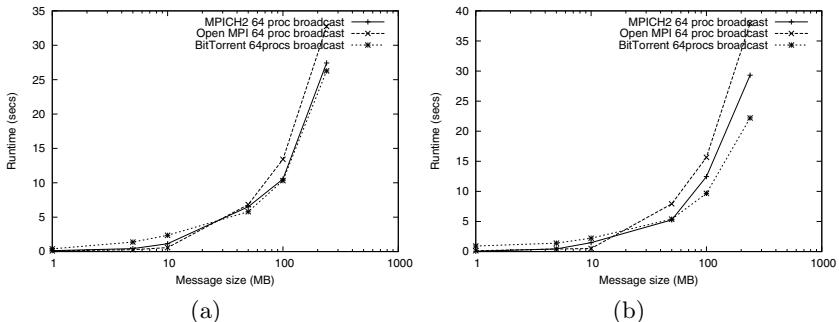


Fig. 4. 64 node broadcasts with main bottleneck link: using clusters Bordeplage and Bordereau (a) or all clusters (Bordeplage, Bordereau and Borderline) (b). This setting has some level of heterogeneity, and BitTorrent performs better than MPI for large messages.

3.6 Interpreting the Results

The performance of the linear tree algorithm of Open MPI decreases significantly with the introduction of a bottleneck link; the throughput decreases by around 50 %. On the other hand, the binomial tree and particularly the BitTorrent algorithm perform excellent. The total amount of received data at all processes does not differ between the different broadcast algorithms. However, the flow of data differs between the three algorithms. Tree-based algorithms statically schedule the transfer of data – e.g. the linear tree algorithm only transfers the exact message size once across the bottleneck link. On the other hand, as described in Sec. 2.2, BitTorrent can use a larger number of parallel point-to-point connections to dynamically schedule the broadcast. This allows the protocol to

utilize the network better. A more detailed analysis confirms that data is broadcast in different ways for BitTorrent and MPI. In Fig. 5, we display a "view" of a 239 MB broadcast with Open MPI and BitTorrent. The dynamics of data movement in a broadcast is difficult to visualize. We choose a view representing the amount of data passing through all switches in any direction during the broadcast. For this purpose we do not use monitoring on the switch level, but a different approach depending on the library we use. For MPI, we know the underlying broadcast algorithm (linear tree) and placement of processes, and we can determine the data movement a-priori. For BitTorrent, as explained in this work, this is not possible, so we measure traffic through additional profiling at each peer instead. Fig. 5 displays the BitTorrent switch traffic in rectangles, and the MPI switch traffic in ellipses. It reveals that inter-cluster communication is more intense with BitTorrent than with MPI – a typical 239 MB broadcast with the setting of Fig. 4(b) transfers around 3.4 GB across the bottleneck switch (in any direction) with BitTorrent, and only the minimal 239 MB with Open MPI. Within clusters however, data exchange with BitTorrent is less intense than with the linear tree algorithm. This behaviour can be partially explained with the fact that BitTorrent is topology-unaware. However, the protocol observes topology to some extent, since the intra-cluster communication is still more intense than inter-cluster communication. Fig. 5 suggests that rather than ignore topology, BitTorrent uses a different ratio between intra- and inter-cluster communication. We conclude that in the presence of bottleneck links, BitTorrent dynamically finds a more efficient schedule for a broadcast than a good tree-based algorithm.

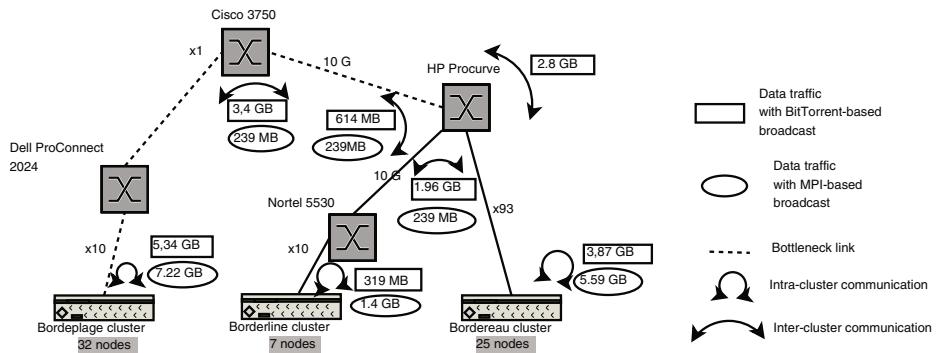


Fig. 5. Data flow in a 239 MB broadcast with BitTorrent. Traffic along switches is shown in rectangles for BitTorrent, and in ellipses for Open MPI.

4 Conclusion

In this work, we compared a BitTorrent broadcast with state-of-the-art MPI broadcasts on switched Ethernet clusters which are either homogeneous or introduce some level of heterogeneity through bottleneck links. The results demonstrate that while in the former setting the linear tree broadcast of Open MPI is

more efficient, for the somewhat heterogeneous setting with a bottleneck link the BitTorrent broadcast outperforms both MPI implementations for large enough messages. We concluded that the intense point-to-point communication in BitTorrent and the resulting adaptive and dynamic schedule of a broadcast can lead to a better performance.

The setting we use is significantly closer to the high-performance computing domain than any previous experimental work using BitTorrent. This means that large-message broadcasts with BitTorrent should be considered in this domain even for moderately heterogeneous networks, and should be preferred for more heterogeneous networks.

Furthermore, the BitTorrent protocol in its tested version is oblivious of the network topology. In a common case in grid and cloud infrastructures, where the network topology is unknown a priori to the user, the BitTorrent protocol is a more sensible choice than MPI for large message broadcasts. If we run applications broadcasting large messages on homogeneous clusters, BitTorrent will be nearly as efficient as modern MPI implementations. If such applications are run on clusters with some level of network heterogeneity, the BitTorrent broadcast will outperform its MPI counterparts.

We refer to our recent work [6], which describes in further detail how BitTorrent's dynamics and adaptivity can be used also in another context – namely for network discovery in grids and clouds.

Acknowledgment. We are thankful to the technical staff at the Bordeaux site of Grid'5000 for the information on the underlying network.

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/IN.1/I2054.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see <https://www.grid5000.fr>).

References

1. Beaumont, O., Marchal, L., Robert, Y.: Broadcast trees for heterogeneous platforms. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, p. 80b (April 2005)
2. Bhat, P., Raghavendra, C., Prasanna, V.: Efficient collective communication in distributed heterogeneous systems. In: Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, pp. 15–24 (1999)
3. den Burger, M.: High-throughput multicast communication for grid applications. Ph.D. thesis, Vrije Universiteit Amsterdam (2009)
4. den Burger, M., Kielmann, T.: Collective receiver-initiated multicast for grid applications. *IEEE Transactions on Parallel and Distributed Systems* 22(2), 231–244 (2011)
5. Cohen, B.: Incentives build robustness in BitTorrent (2003)
6. Dichev, K., Reid, F., Lastovetsky, A.: Efficient and reliable network tomography in heterogeneous networks using BitTorrent broadcasts and clustering algorithms. In: SC 2012 (2012)

7. Dichev, K., Rychkov, V.M., Lastovetsky, A.: Two Algorithms of Irregular Scatter/Gather Operations for Heterogeneous Platforms. In: Keller, R., Gabriel, E., Resch, M., Dongarra, J. (eds.) EuroMPI 2010. LNCS (LNAI), vol. 2536, pp. 289–293. Springer, Heidelberg (2010)
8. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J. (eds.) EuroPVM/MPI 2004. LNCS, vol. 3241, pp. 97–104. Springer, Heidelberg (2004)
9. Hatta, J., Shibusawa, S.: Scheduling algorithms for efficient gather operations in distributed heterogeneous systems. In: Proc. Int Parallel Processing Workshops, pp. 173–180 (2000)
10. Izal, M., Urvoy-Keller, G., Biersack, E.W., Felber, P., Al Hamra, A., Garcés-Erice, L.: Dissecting BitTorrent: Five Months in a Torrent’s Lifetime. In: Barakat, C., Pratt, I. (eds.) PAM 2004. LNCS, vol. 3015, pp. 1–11. Springer, Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-24668-8_1
11. Keller, R., Bosilca, G., Fagg, G., Resch, M., Dongarra, J.: Implementation and Usage of the PERUSE-Interface in Open MPI. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 347–355. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11846802_48
12. Kielmann, T., Bal, H.E., Gorlatch, S.: Bandwidth-efficient collective communication for clustered wide area systems. In: Proc. 14th Int. Parallel and Distributed Processing Symp., IPDPS 2000, pp. 492–499 (2000)
13. Patarasuk, P., Faraj, A., Yuan, X.: Pipelined broadcast on Ethernet switched clusters. In: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, p. 10 (April 2006)
14. Rodriguez, P., Biersack, E.W.: Dynamic parallel access to replicated content in the Internet. IEEE/ACM Trans. Netw. 10, 455–465 (2002), <http://dx.doi.org/10.1109/TNET.2002.801413>
15. Wadsworth, D.M., Chen, Z.: Performance of MPI broadcast algorithms. In: Proc. IEEE Int. Symp. Parallel and Distributed Processing, IPDPS 2008, pp. 1–7 (2008)

MIP Model Scheduling for Multi-Clusters

Héctor Blanco, Fernando Guirado, Josep Lluís Lérida, and V.M. Albornoz

Universitat de Lleida, Universidad Técnica Federico Santamaría
`{hectorblanco,f.guirado,jlerida}@diei.udl.cat,`
`victor.albornoz@usm.cl`

Abstract. Multi-cluster environments are composed of multiple clusters that act collaboratively, thus allowing computational problems that require more resources than those available in a single cluster to be treated. However, the degree of complexity of the scheduling process is greatly increased by the resources heterogeneity and the co-allocation process, which distributes the tasks of parallel jobs across cluster boundaries.

In this paper, the authors propose a new MIP model which determines the best scheduling for all the jobs in the queue, identifying their resource allocation and its execution order to minimize the overall makespan. The results show that the proposed technique produces a highly compact scheduling of the jobs, producing better resources utilization and lower overall makespan. This makes the proposed technique especially useful for environments dealing with limited resources and large applications.

Keywords: Job Scheduling, Multi-Cluster, Co-Allocation, MIP Model.

1 Introduction

Computation problems that require the use of a large amount of processing resources can be solved by the use of multiple clusters in a collaborative manner. These environments, known as multi-clusters, are distinguished from grids by their use of dedicated interconnection networks [1].

In those environments the scheduler has access to distributed resources across different clusters to allocate those jobs that cannot be assigned to a single cluster [2]. This allocation strategy, known as co-allocation, can maximize the job throughput by reducing the queue waiting times, and thus, jobs that would otherwise wait in the queue for local resources can begin its execution earlier, improving system utilization and reducing average queue waiting time [2]. However, mapping jobs across the cluster boundaries can result in rather poor overall performance when co-allocated jobs contend for inter-cluster network bandwidth. Additionally, the heterogeneity of processing and communication resources increases the complexity of the scheduling problem [3].

It is possible to find in the literature multiple studies based on co-allocation. In [2] different scheduling strategies using co-allocation were analyzed, concluding that unrestricted co-allocation is not recommendable. Other studies have dealt with co-allocation by developing load-balancing techniques [4], selecting the most

powerful processors [5] or minimizing the inter-cluster link usage [3] without finding a compromise between them. In [6] an analytical model was presented in order to reduce the parallel jobs execution time by considering both resource availability: processors and communication.

A common issue in previous works is that jobs are allocated individually. Thus, allocating the best available resources to a job without considering the requirements of the rest of jobs present in the waiting queue, can reduce the performance of future allocations and the overall system performance [7]. To solve this, in [8] the authors presented a scheduling strategy based on a linear programming model, which brings together the parallel jobs in the waiting queue that *fit* the available resources and allocates them simultaneously.

The main constraint on this strategy is the limitation to the set of jobs that fit on the available resources. In the present work, the authors proposed a new MIP model referenced as *OAS* for Ordering and Allocation Scheduling, capable of determine the best resources allocation and the order in which they must be executed. Although the scheduling problem is NP-hard, the results shown that by considering sets of jobs it is possible to improve the resources utilization and the overall performance.

2 Related Work

In the literature there are multiple strategies for the scheduling process that can be grouped into two categories; on-line and off-line modes. In the on-line mode, only arrived jobs to the system are known, and the allocation decisions are restricted to those jobs. On the other hand, the off-line mode has knowledge of all the jobs considering the whole set of job for allocation decisions [9]. Nevertheless, in on-line systems with high job-interarrival rates, the scheduling problem can be addressed with job-clustering techniques to improve the overall performance, the system utilization, etc.

The on-line techniques allocate only one job without taking into account the rest of the waiting jobs, loosing relevant information to improve the overall system performance. By maintaining the job arrival order, resources that are available may end up not being allocated. The backfilling technique aims to solve this allowing to be moved up smaller jobs from the back of the queue [10]. Shmueli et al. proposed a look-ahead optimizing scheduler to generate the local optimal backfill selection by using dynamic programming [7]. Shah et al. proposed a near optimal job packing algorithm to reduce the chances of job killing and minimize external fragmentation [11]. These approaches tried to map only the jobs that better fill the gaps without considering other packing opportunities. Previous research has shown that slightly modifying the execution order of jobs can improve utilization and offer new optimization opportunities [7][8].

The strategies previously presented are extensively used on Parallel machines and Cluster computing environments. Nevertheless, they are based on specific environment characteristics and in most cases assuming jobs with independent tasks, i.e, without communication constraints. In the present paper, we consider

jobs with a fixed number of processors requirement, known as rigid Bulk-Synchronous Parallel jobs [7]. The meta-scheduling on multi-cluster resources is more challenging than traditional single-domain systems, due to the heterogeneous dynamic resources availability in different administrative domains, and the continuous arrival of jobs at the meta-scheduler [12]. Hence, to face the new challenges on multi-cluster environments new heuristics should be proposed.

The research on multi-cluster and grid environments has provided multiple scheduling solutions based on different criteria: cost, makespan, etc. Feng et al. [13] proposed a deadline cost optimization model for scheduling one job. Buyya et al. [14] proposed a greedy approach with deadline and cost constraints for efficient deployment of an individual job. In contrast, the current paper is focused on the concurrent scheduling of many jobs. In [15][16] heuristics and Genetic Algorithm solutions for scheduling concurrent jobs are proposed. However, those studies assumed independent jobs with no communication restrictions.

3 Ordering and Allocation Scheduling Strategy

System performance have different meanings. Final users performance deal with reducing the job execution time. However, system administrators would like to maximize the resources usage. In an environment with large job inter-arrival time, the allocation mechanism is responsible of improving both performances.

However, in situations with low inter-arrival time, jobs accumulate in the waiting queue generating new scheduling opportunities. In these situations, both execution order and allocation strategy are decisive for improving overall performance. In the present work, a new MIP model (*OAS*), which manages the packing of jobs in the waiting queue to minimize their makespan, thus improving the system utilization and user satisfaction is proposed. In order to do that, *OAS* deals with two challenges: (i) resources heterogeneity and availability and (ii) tasks from a job can be assigned to different clusters in a co-allocation process. In these circumstances, the allocation not only has to consider the processing time, but also the communication capabilities in order to avoid inter-cluster link saturation, which could produce unpredictable effects on job performance.

3.1 Problem Statement

Multi-Cluster Model. A multicluster environment is made up of a set of α arbitrary sized clusters with heterogeneous resources. Let $M = \{C_1..C_\alpha\}$ is the set of Cluster sites; $R = \{R_1^1, R_2^1..R_{n-1}^\alpha, R_n^\alpha\}$ the set of processing resources of the multi-cluster, being n the total number of nodes. Each cluster is connected to each other by a dedicated link through a central switch. Let $\mathcal{L} = \{\mathcal{L}_1..\mathcal{L}_\alpha\}$ the inter-cluster links being \mathcal{L}_k the link between the site C_k and the central switch.

The processing resources capabilities are represented by the Effective Power metric (Γ^r) defined in [6]. This normalized metric relates the processing power of each resource with its availability. Thus, $\Gamma^r = 1$ when processing resource $r \in R$ has capacity to run tasks at full speed, and otherwise $\Gamma^r < 1$.

Parallel Application Job Model. In this work, we consider parallel application jobs with a fixed number of processing resources requirements known as rigid parallel jobs [7]. A job j is composed of a fixed number of tasks that act in a collaborative manner. Each task τ_j is comprised of various processing, communication and synchronization phases. In our case, each job task uses an all-to-all communication pattern with similar processing and communicating requirements, where all tasks start and finish at the same time, following the Bulk-Synchronous Parallel model. Job assignment is static avoiding re-allocations while the job is being executed. Additionally, jobs can be co-allocated on different clusters in order reduce its execution time.

Thus, the estimated execution time for the parallel job j can be modeled by

$$Te_j = Tb_j \cdot ct_j, \quad \forall j \in J \quad (1)$$

where Tb_j is the base time of j in dedicated resources and ct_j is the time cost factor when job is allocated on resources S . The base-time Tb_j is assumed to be known based on user-supplied information, experimental data, job profiling, etc.

Time Cost Model. In the literature, the time cost ct_j is obtained from the allocated resources without considering communications [3], or considering a fixed communications penalty when co-allocation is applied [17]. In contrast, we modeled the time cost based on heterogeneity of the selected processing resources and the availability of the inter-cluster links used, expressed by

$$ct_j = \sigma_j \cdot SP_j + (1 - \sigma_j) \cdot SC_j, \quad \forall j \in J \quad (2)$$

where SP_j denotes the processing slowdown from resources, SC_j is the communication slowdown by inter-cluster links, and σ_j is the processing time relevance with respect to the communication time. The σ_j is obtained by characterizing the job as the base-time Tb_j . The SP_j factor is obtained from the slowest processing resource, i.e. which provides the maximum processing slowdown,

$$SP_j = \max_{\forall r \in S} \{SP_j^r\}, \quad \forall j \in J \quad (3)$$

where SP_j^r is the processing slowdown from the effective power of resource r .

SC_j evaluates the communication slowdown by inter-cluster link contention. The co-allocation of a parallel job application consumes a certain amount of bandwidth in each inter-cluster link \mathcal{L}_k , denoted by BW_j^k , and calculated by

$$BW_j^k = (t_j^k \cdot PTBW_j) \cdot \left(\frac{\tau_j - t_j^k}{\tau_j - 1} \right), \quad \forall k \in 1 \dots \alpha, j \in J \quad (4)$$

where $PTBW_j$ is the required per-task bandwidth, τ_j is the total number of tasks of j and t_j^k is the number of those allocated on cluster C_k . The first term in the equation is the total bandwidth consumed by tasks on cluster C_k , while the second is the percentage of communication with other clusters.

When co-allocated jobs use more bandwidth than the available, saturation occurs, and jobs sharing this link are penalized increasing their communication time. The inter-cluster links saturation degree relates the maximum bandwidth of each link with the bandwidth requirements of the allocated parallel jobs

$$SAT^k = \frac{MBW}{\sum_{\forall j} (BW_j^k)}, \quad k \in 1 \dots \alpha, j \in J \quad (5)$$

when $SAT^k \geq 1$ the link \mathcal{L}_k is not saturated, otherwise is saturated delaying the jobs that use it, with a slowdown expressed by

$$SC_j^k = \begin{cases} (SAT^k)^{-1} & \text{when } SAT^k < 1 \\ 1 & \text{otherwise} \end{cases} \quad (6)$$

Communication slowdown SC_j comes from the most saturated link used

$$SC_j = \max_{\forall k} \{SC_j^k, \quad \forall j \in J\}, \quad (7)$$

Allocation and Scheduling Mechanism. The most common scheduling techniques allocate the jobs separately, without taking into account the requirements of further jobs. This is represented in Figure 1(a) where a *First Come First Served* scheduling has been applied. Better performance results can be achieved by the set of jobs together, as can be seen in Figure 1(b).

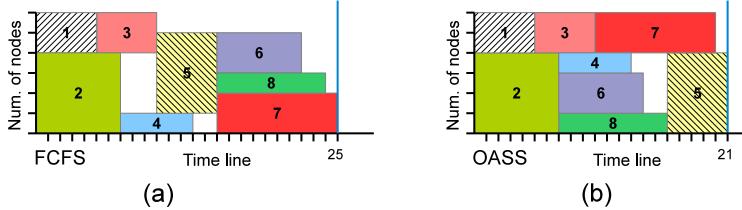


Fig. 1. Job scheduling: (a) FCFS allocation, (b) allocation grouping tasks

3.2 The Mixed-Integer Programming Model

Mixed-Integer Programming (MIP) allows to obtain the solutions that maximize or minimize an objective function under some constraints. In this paper, the objective function is the makespan, thus the obtained solution provides the allocation and execution order for each treated job that reduce their execution time, also minimizing the resources idle time. In [17] was determined that in some situations a certain threshold on the saturation degree may be allowed. For simplicity, in the present study, we restrict the model to those solutions that avoid the saturation, by evaluating the inter-cluster links usage. Next we present the MIP model shown in Figure 2.

Parameters and Variables. First, the multi-cluster environment is described; the set of resources R and their effective power (Γ^r), inter-cluster links (\mathcal{L}) and the maximum available bandwidth for each inter-cluster link $k \in \mathcal{L}$ (MBW_k). Next, for each job j : the number of tasks (τ_j), its base-time (Tb_j), the required per-task bandwidth ($PTBW_j$) and the weighting factor (σ_j), which measures the relevance of the processing and communication time (lines 1-9).

The decision variables define the job order and allocation (lines 13-20). The allocation is expressed by a binary variable, $Z_{(j,r)} = 1$ (line 13) when the job j is assigned to r and 0 otherwise. To obtain the job execution order, the allocation

domain of each resource is splitted into time-slots (lines 14-17), being $X_{(j,r,t)} = 1$ when job j is assigned to resource r in the time-slot t . Let $T = \{T^1..T^\theta\}$ be the set of time-slots and θ the total number of time-slots used by the set of jobs. Variable $Y_{(j,t)}$ is set if job j starts its execution in the time-slot t (line 15).

Input Parameters

1. R : Set of processing resources.
2. \mathcal{L} : Set of inter-cluster links.
3. I^r : Effective power for resource r , $\forall r \in R$
4. MBW_k : Maximum Available bandwidth for each inter-cluster link k , $\forall k \in \mathcal{L}$.
5. J : set of jobs to be allocated.
6. τ_j : number of tasks making up job j , $\forall j \in J$.
7. Tb_j : execution base-time for the job j , $\forall j \in J$.
8. $PTBW_j$: required bandwidth for each jobs task, $\forall j \in J$
9. σ_j : time-processing and -communication weighting factor, $\forall j \in J$.
10. T : Set of time-slots in which job can be assigned.
11. θ : total number of time-slots. Deadline for the set of jobs.
12. η : time-slot size.

Variables

13. $Z_{(j,r)} = 1$ if j is assigned to resource r , $\forall j \in J, r \in R$
14. $X_{(j,r,t)} = 1$ if j is assigned to resource r in slot t , $\forall j \in J, r \in R, t \in T$
15. $Y_{(j,t)} = 1$ if j starts running in slot t , $\forall j \in J, t \in T$
16. s_j : time-slot in which job j starts running, $\forall j \in J$
17. f_j : time-slot in which job j is completed, $\forall j \in J$
18. SP_j is the processing slowdown of job j , $\forall j \in J$
19. $BW_{j,k,t}$: Bandwidth consumed by job j on link k in slot t . $\forall j \in J, k \in \mathcal{L}, t \in T$
20. $ABW_{k,t}$: Available bandwidth on link k , in slot t , $\forall k \in \mathcal{L}, t \in T$

Objective function

21. Minimize the makespan of the set of jobs

Fig. 2. MIP model representation

Based on the time-slot, the job j execution time is determined by its starting time-slot, s_j , and the last used time-slot, f_j . The time-slots occupied by j are calculated considering the job base-time (Tb_j), the processing slowdown (SP_j) that the allocated resources provide and the time-slot size (η), expressed by

$$f_j = s_j + (Tb_j * (SP_j * \sigma_j + (1 - \sigma_j))) / \eta \quad (8)$$

Processing slowdown SP_j (line 18) is calculated by equ. 3. The communication slowdown is not considered because the solutions are those that avoid the saturation of inter-cluster links (lines 19-20). Variable $BW_{j,k,t}$ is the bandwidth consumed on the inter-cluster link k for the time-slot t , and $ABW_{k,t}$ is the available bandwidth on link k on time-slot t once all jobs have been allocated.

Objective Function. When there are many possible solutions, the objective function defines the quality of each feasible solution. The aim of the model is to minimize the global makespan that is determined by the latest completed job

$$\text{minimize} \{ \max_{\forall j \in J} (f_j) \} \quad (9)$$

Constraints. The constraints contribute to define the correct solutions to the problem. Thus, the model must ensure that all tasks from a parallel job are allocated and start at the same time. We define the following set of constraints:

$$\sum_{\forall j \in J} X_{(j,r,t)} \leq 1, \quad \forall r, t \quad (10)$$

$$Z_{(j,r)} = \max_{t \in T} (X_{j,r,t}), \quad \forall j, r \quad (11)$$

$$\sum_{\forall r \in R} Z_{(j,r)} = \tau_j, \quad \forall j \quad (12)$$

$$\sum_{\forall r' \in R} (X_{j,r',t}) \geq (\tau_j \cdot X_{j,r,t}), \quad \forall j, r, t \quad (13)$$

$$X_{(j,r,t')} \leq 1 - Y_{(j,t)} \quad \forall j, r \wedge t' \in 1..(t-1) \quad (14)$$

$$\sum_{\forall t \in T} Y_{(j,t)} = 1 \quad \forall j \quad (15)$$

$$s_j = \sum_{\forall t \in T} (Y_{j,t} \cdot t) - 1 \quad \forall j \quad (16)$$

$$f_j = \max_{\forall r \in R, t \in T} (X_{(j,r,t)} \cdot t) \quad \forall j \quad (17)$$

$$ct_j = \sigma_j \cdot SP_j + (1 - \sigma_j) \quad \forall j \quad (18)$$

$$(f_j - s_j) \cdot \eta \leq (Tb_j \cdot ct_j) \quad \forall j \quad (19)$$

$$\left(\sum_{\forall r \in R, t \in T} X_{j,r,t} \cdot \eta \right) \geq (Tb_j \cdot ct_j \cdot \tau_j) \quad \forall j \quad (20)$$

$$ABW_{k,t} = MBW_k - \sum_{\forall j \in J} BW_{j,k,t} \quad \forall j, k, t \quad (21)$$

$$ABW_{k,t} \geq 0 \quad \forall k, t \quad (22)$$

$$Z_{(j,r)} \in \{0, 1\}, \quad X_{j,r,t} \in \{0, 1\}, \quad Y_{(j,t)} \in \{0, 1\},$$

$$s_j \geq 0, \quad f_j \geq 0, \quad ct_j \geq 0 \quad (23)$$

Constraint set (10) ensures that a resource can only be allocated to a task simultaneously. Constraint set (11) defines the variable $Z_{(j,r)}$, equals to 1 when job j is allocated to the resource r and 0 otherwise. Constraint set (12) ensures that all tasks τ_j are allocated. Constraint set (13) guarantees that all the tasks of a job are executed at the same time but in different resources. Constraint sets (14) and (15) define the variable $Y_{(j,t)}$, equals to 1 when the j th job initiates its execution in the t th time-slot, otherwise is 0. In constraint (16) and (17) the variables s_j and f_j are defined as the starting and finishing time-slot for the j th job. Variable s_j is obtained from variable $Y_{(j,t)}$ and variable f_j is calculated considering the slowest allocated resource. In constraint set (18), the execution cost ct_j the processing slowdown is obtained from the slowest allocated resource. Constraint sets (19) and (20) ensure that the time-slots are contiguous and in accordance with the time spent on the slowest allocated resource. Constraint set (21) calculates the available bandwidth of the k th inter-cluster link in the t th time-slot, $ABW_{k,t}$. Finally, constraint set (22) guarantees non-saturation of the inter-cluster links in every time-slot.

4 Experimentation

The experimental study was carried out in order to: (1) evaluate the influence of the time-slot duration size on the scheduling solutions and (2) determine the effectiveness of the scheduling solutions provided by *OAS*. The first study used a synthetic workload varying the size of the time-slot. On the second, *OAS* was compared with heuristics from the literature.

OAS was implemented by using the *CPLEX* linear mixed integer programming solver and the solutions were applied on the GridSim simulation framework, characterized as a heterogeneous multi-cluster system. The scheduling of applications with independent tasks is NP-Complete, and MIP models are known to be very computational demanding. Due to this, the environment was limited to 3 clusters, each one with 4 nodes interconnected by a Gigabit network. The heterogeneity was defined by different effective power to each cluster with values of $\Gamma_k = \{1.0, 0.75, 0.5\}$ respectively.

4.1 Time-Slot Size Analysis

To determine the job execution order, *OAS* uses the time-slot concept. The time-slot size could limit the quality of the scheduling solution, so a study was performed to evaluate the impact of the time-slot size on the results. We defined a workload composed by 8 jobs, with high computational requirements ($\sigma_j = 0.7$), with an average job base time of 67×10^4 seconds, with different number of tasks, from 1 to 12 and with different computation and communication requirements, representative of parallel jobs in the fields of weather prediction, fluid simulation, etc. identified by their very large computational requirements.

The time-slot values were in the range {15%..250%} times the average of the jobs base-time from the workload. Figure 3 shows the makespan and the solver execution time for each case of study. As can be seen, when the slot size increases, the makespan increases. This is because bigger slots reduces the resources availability during long periods of time even allocated jobs have finished. Thus, the start time of upcoming jobs is delayed until the time-slot finishes.

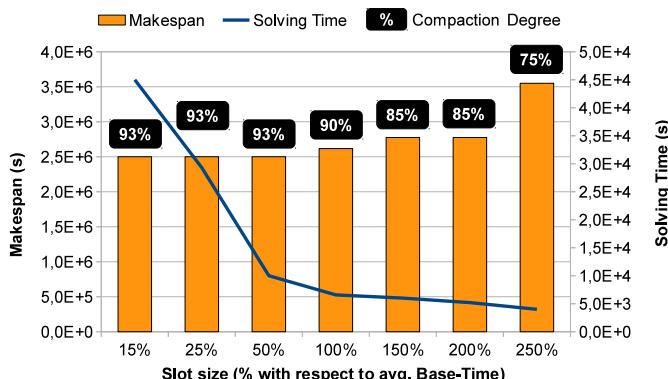


Fig. 3. Comparison for the time-slot size

The shortest the time-slot is, the better makespan is obtained. However, the model becomes more complex increasing the solving time. From sizes lower than 50%, makespan values become similar to the minimum makespan. In the other way the solving time grows up exponentially. Figure 3, also show the compaction degree, which measures the percentage of time in which the resources are been used respect the overall execution time. Thus, the higher the compaction degree is, the more exploited and less idle the resources are.

4.2 OAS Performance Evaluation

To determine the effectiveness of the solutions provided by *OAS*, we compare the results with some of the most common scheduling techniques in the literature: *First Come First Served* (FCFS), *Short Jobs First* (SJF), *Big Jobs First* (BJF), *Fit Processors First Served* (FPFS), *Short Processing Time* (SPT) and *Long Processing Time* (LPT).

In this experimental study we defined a set of six synthetic workloads composed by 8 jobs with similar characteristics of computational requirements as in the previous experimental study. We defined two of them in order to fit well a specific scheduling technique and thus limiting any solver advantage. Then, the first workload *WL-1* was designed to perform well with the techniques that try to match the available resources with the processing requirements. *WL-2* workload was designed to perform well with the techniques that prioritize smaller jobs. Finally, workloads *WL-3* to *WL-6*, designed without taking any particular criteria into account. The metrics used were the makespan (Figure 4(a)) and the compaction degree (Figure 4(b)). The time-slot size was defined to the 50% of the average job base time.

It can be observed that each technique has a different behavior. The technique that in some workload performs well in another obtains worst makespan and lower compaction degree. But in all cases *OAS* obtained good makespan and compaction results, irrespectively on the workload nature. This is due to its ability to have a global vision on the resources requirements for the whole set of jobs and their availability. Thus, by defining the correct job execution order and task to resource

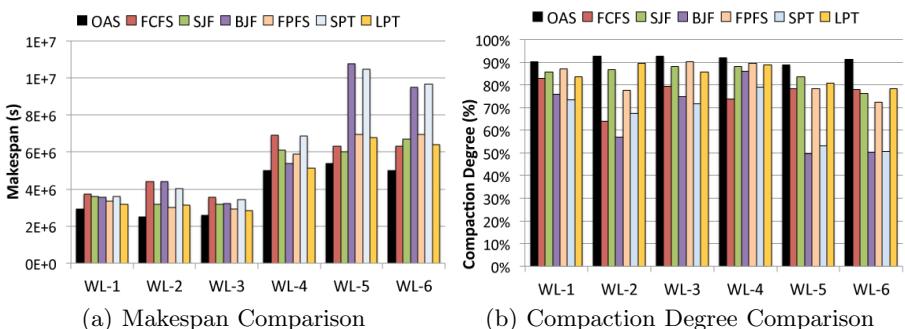


Fig. 4. Comparison of six kinds of workloads

allocation, it is possible to obtain the better scheduling decisions that reduces the makespan, increasing the compaction degree and improving the resources usage.

5 Conclusions

In the present work, we focused on the scheduling process of BSP parallel applications on heterogeneous multi-cluster environments, by applying multiple job allocation and co-allocation when it is necessary. The goal is to determine the effectiveness of the job execution order and the multiple-job allocation with processing and communication resource considerations, and thus, a *Mixed-Integer Programming* model had been developed. The results were compared with other scheduling techniques from the literature, obtaining better makespan results and resources usage. However, the presented MIP model has a great computational complexity. By this, we are developing a low complexity scheduling heuristic with similar goals.

Acknowledgment. This work was supported by the MECS of Spain under contract TIN2011-28689-C02 and the CUR of DIUE of GENCAT and the European Social Fund. Also, DGIP (Grant USM 28.10.37) and CIDIEN of Univ. Técnica Federico Santa María.

References

1. Javadi, B., Akbari, M.K., Abawajy, J.H.: A performance Model for Analysis of Heterogeneous Multi-Cluster Systems. *Parallel Computing* 32(11-12), 831–851 (2006)
2. Bucur, A.I.D., Epema, D.H.J.: Schedulling Policies for Processor Coallocation in Multicluster Systems. *IEEE TPDS* 18(7), 958–972 (2007)
3. Jones, W., Ligon, W., Pang, L., Stanzione, D.: Characterization of Bandwidth-Aware Meta-Schedulers for Co-Allocating Jobs Across Multiple Clusters. *Journal of Supercomputing* 34(2), 135–163 (2005)
4. Yang, C., Tung, H., Chou, K., Chu, W.: Well-Balanced Allocation Strategy for Multiple-Cluster Computing. In: *IEEE Int. Conf. FTDCS 2008*, pp. 178–184 (2008)
5. Naik, V.K., Liu, C., Yang, L., Wagner, J.: Online Resource Matching for Heterogeneous Grid Environments. In: *Int. Conf. CCGRID 2005*, vol. 2, pp. 607–614 (2005)
6. Lérida, J.L., Solsona, F., Giné, F., García, J.R., Hernández, P.: Resource Matching in Non-dedicated Multicluster Environments. In: Palma, J.M.L.M., Amestoy, P.R., Daydé, M., Mattoso, M., Lopes, J.C. (eds.) *VECPAR 2008. LNCS*, vol. 5336, pp. 160–173. Springer, Heidelberg (2008)
7. Shmueli, E., Feitelson, D.G.: Backfilling with Lookahead to Optimize the Packing of Parallel Jobs. *J. Parallel Distrib. Comput.* 65(9), 1090–1107 (2005)
8. Blanco, H., Lérida, J.L., Cores, F., Guirado, F.: Multiple Job Co-Allocation Strategy for Heterogeneous Multi-Cluster Systems Based on Linear Programming. *Journal of Supercomputing* 58(3), 394–402 (2011)
9. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U.: Parallel Job Scheduling — A Status Report. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) *JSSPP 2004. LNCS*, vol. 3277, pp. 1–16. Springer, Heidelberg (2005)

10. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. *IEEE TPDS* 18(6), 789–803 (2007)
11. Hussain, S., Qureshi, K.: Optimal job packing, a backfill scheduling optimization for a cluster of workstations. *Journal of Supercomputing* 54(3), 381–399 (2010)
12. Zhang, W., Cheng, A.M.K., Hu, M.: Multisite co-allocation algorithms for computational grid. In: IPDPS 2006, pp. 335–335 (2006)
13. Feng, H., Song, G., Zheng, Y., Xia, J.: A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Dependent Tasks in Grid Computing. In: Li, M., Sun, X.-H., Deng, Q., Ni, J. (eds.) *GCC 2003. LNCS*, vol. 3033, pp. 113–120. Springer, Heidelberg (2004)
14. Buyya, R., Murshed, M., Abramson, D., Venugopal, S.: Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm. *Softw. Pract. Exper.* 35(5), 491–512 (2005)
15. Munir, E.U., Li, J., Shi, S.: QoS sufferage heuristic for independent task scheduling in grid. *Information Technology Journal* 6(8), 1166–1170 (2007)
16. Garg, S., Buyya, R., Siegel, H.: Time and cost trade-off management for scheduling parallel applications on Utility Grids. *Future Generation Computer Systems* 26(8), 1344–1355 (2010)
17. Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., Streit, A.: On Advantages of Grid Computing for Parallel Job Scheduling. In: Int. Conf. CCGRID 2002 (2002)
18. Li, H., Groep, D., Wolters, L.: Workload Characteristics of a Multi-cluster Supercomputer. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) *JSSPP 2004. LNCS*, vol. 3277, pp. 176–193. Springer, Heidelberg (2005)

Systematic Approach in Optimizing Numerical Memory-Bound Kernels on GPU

Ahmad Abdelfattah¹, David Keyes¹, and Hatem Ltaief²

¹ Division of Mathematical and Computer Sciences and Engineering

² Supercomputing Laboratory

King Abdullah University of Science and Technology

Thuwal, Saudi Arabia

Abstract. The use of GPUs has been very beneficial in accelerating dense linear algebra computational kernels (DLA). Many high performance numerical libraries like CUBLAS, MAGMA, and CULA provide BLAS and LAPACK implementations on GPUs as well as hybrid computations involving both, CPUs and GPUs. GPUs usually score better performance than CPUs for compute-bound operations, especially those characterized by a regular data access pattern. This paper highlights a systematic approach for efficiently implementing memory-bound DLA kernels on GPUs, by taking advantage of the underlying device's architecture (e.g., high throughput). This methodology proved to outperform existing state-of-the-art GPU implementations for the symmetric matrix-vector multiplication (SYMV), characterized by an irregular data access pattern, in a recent work (Abdelfattah et. al., VECPAR 2012). We propose to extend this methodology to the general matrix-vector multiplication (GEMV) kernel. The performance results show that our GEMV implementation achieves better performance for relatively small to medium matrix sizes, making it very influential in calculating the Hessenberg and bidiagonal reductions of general matrices (radar applications), which are the first step toward computing eigenvalues and singular values, respectively. Considering small and medium size matrices (≤ 4500), our GEMV kernel achieves an average 60% improvement in single precision (SP) and an average 25% in double precision (DP) over existing open-source and commercial software solutions. These results improve reduction algorithms for both small and large matrices. The improved GEMV performances engender an average 30% (SP) and 15% (DP) in Hessenberg reduction and up to 25% (SP) and 14% (DP) improvement for the bidiagonal reduction over the implementation provided by CUBLAS 5.0.

Keywords: Matrix-Vector Multiplication, GPU Optimizations, Memory-Bound Operations, Hessenberg Reduction, Bidiagonal Reduction.

1 Introduction

The high level of parallelism found on modern GPUs has attracted the scientific community to think about porting their legacy applications on GPUs. Since then, GPUs have been one of the favorite choices for accelerating many computational kernels, especially

when it comes to kernels with regular data access patterns. Considering only NVIDIA GPUs, they are usually equipped with hundreds of lightweight floating point (CUDA) cores, grouped into streaming multiprocessors (SMs). Starting from Fermi, each SM has a private parallel L1-cache/shared memory. All SMs share L2-cache and the global DRAM. Although the peak performance of modern GPUs has already reached the terascale, developers have to pay attention to several paramount parameters in order to achieve decent performance. For example, ensuring coalesced memory accesses, reducing shared bank conflicts, avoiding register spilling, and removing synchronization points are all performance optimizations that have direct impact on the overall GPU kernel quality. However, kernels that are memory-bound by nature rarely achieve performance close to the theoretical peak. In such kernels, the maximum achievable performance is usually limited by the global memory bandwidth.

This paper highlights a methodology for accessing dense matrices in numerical kernels, applied to the general matrix-vector multiplication (GEMV) kernel as a case study. This methodology was part of a previous work in accelerating the symmetric matrix-vector multiplication (SYMV) kernel, where significant improvement was achieved over state-of-the-art designs. Since both SYMV and GEMV are Level 2 BLAS operations, hence bounded by the bus memory throughput, we extended the same methodology to the GEMV kernel. The proposed way of accessing the matrix assumes that it is divided into square blocks. Each block is read from global memory using a simple double buffering technique combined with large occupancy to hide the latency of the stalled warps due to memory loads. Our experimental results against the latest CUBLAS 5.0 [6], CULA dense library [1] and MAGMA 1.2 [2] show improvements for small to medium matrix sizes, especially for the non-transposed case. The new design still maintains roughly the same performance for large matrix sizes. The GEMV enhancements obtained for these matrix sizes engender a significant speedup when plugged into the Hessenberg and bidiagonal reduction drivers, in which the GEMV variants (transpose and non-transpose) represent the fundamental operations. Both reductions correspond to the first step toward calculating the eigenvalues and singular values. Noteworthy to mention that radar detection applications [11,15] require the computation of several independent small eigenvalue and singular value decomposition problems.

The remainder of the paper is organized as follows: Section 2 presents some related work in optimizing numerical linear algebra for GPUs. In Section 3, we give some information about the work that has been done for the SYMV kernel [7]. Its potential extension to the GEMV kernel as well as the GEMV kernel implementation details are described in Section 4, while the results are shown in Section 5. We give our conclusion and provide suggestions for future work in Section 6.

2 Related Work

There is a lot of work done for accelerating dense linear algebra kernels on GPUs using CUDA. This is very critical as many applications rely on these basic block kernels to extract their actual performance. The latest CUBLAS 5.0 [6] is often the reference implementation for NVIDIA GPU BLAS. MAGMA [2] provides a very optimized subset of BLAS functions (MAGMABLAS) for GPUs and a hybrid implementation

(using both, the host and the device) of the major LAPACK routines. Indeed, it offers some improved implementations over CUBLAS, such as the SYMV [12] kernel and the GEMM kernel [13]. CUBLAS and MAGMA are freely available for public downloads. CULA Dense [1] is a commercial package and represents a set of dense linear algebra libraries developed for GPUs. It is also written in CUDA, and further improves CUBLAS implementations [8]. In this paper, we compare our results against the three aforementioned libraries i.e., CUBLAS, MAGMA, and CULA.

On the other hand, the evolving architecture of modern GPUs pushed the need for exposing tunable parameters or hooks in the kernel implementations so that productivity is maintained. Volkov and Demmel [14] proposed an early benchmarking scheme to tune DLA routines, such as matrix-matrix multiplication. Kurzak et. al [9] developed an autotuning framework called ASTRA, originally designed over Fermi GPUs for tuning also GEMM kernels and its variants. Autotuning using ASTRA has been recently extended to Kepler [10], the newest NVIDIA GPU architecture to date. Such frameworks can be applied to our SYMV and GEMV kernels, in order to benchmark our kernel design on future architectures, without starting from scratch.

3 SYMV Background

Our methodology of processing matrix blocks has been first introduced in [7], where a new SYMV kernel was proposed. The new SYMV kernel is 2.5x faster than CUBLAS 4.0 and 1.3x better than MAGMABLAS. The kernel design was described following two strategies: the *block-level strategy*, governing movements of thread blocks within the matrix, and the *thread-level strategy*, governing thread mapping within a single matrix block. The idea here is to apply both strategies to the GEMV kernel. However, the block-level strategy of the SYMV kernel necessitates an adjustment, since only half of the matrix needs to be processed for the symmetric case. In GEMV, the whole non-symmetric matrix requires to be scanned. Both block-level and thread-level strategies for GEMV are explained in Section 4. Our results [7] showed a direct impact on tridiagonalisation and symmetric eigenvalue computations.

In general, how a better strategy and optimizations can be found? The answer is to have insight through a profiling or tracing tool. From our experience in developing the SYMV kernel [7], the NVIDIA visual profiler [4] along with PAPI [5] gave us rich information about register usage, L1-cache, shared memory, and the DRAM. For example, during development, we detected register spilling through the number of local memory accesses along with the number of 32-bit registers used per thread. Shared memory bank conflicts can help the developer changing the strategy of accessing shared memory. In memory bounds kernels similar to SYMV and GEMV, the matrix should be loaded only once due to the absence of data reuse of matrix elements. The number of global memory loads help judge if coalesced access is satisfied. Generally, profiling CUDA kernels helps the programmer focus on specific parts of the kernel, identified as potential bottlenecks, in order to enhance its behavior.

We show an updated performance chart of our SYMV kernel, for single and double precisions in Figure 3. There are no changes in the design of our original kernel except a slight adjustment to better handle irregular matrix dimensions (dimensions that are

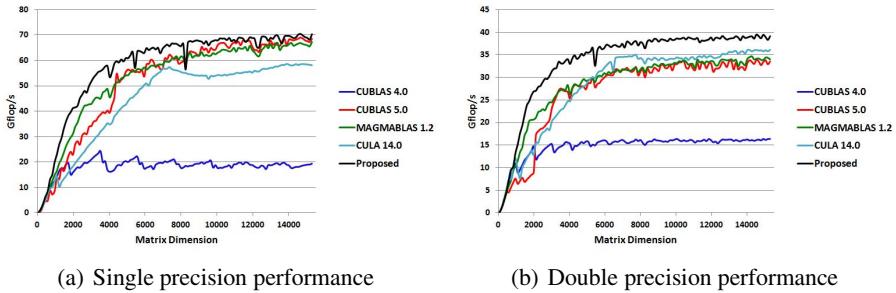


Fig. 1. Performance of SYMV kernel using five different implementations

not multiple of the internal blocking size i.e., 64). However, what is new about the Figure 3 is that it includes performance curves for two other kernel implementations. The first one is the CUBLAS 5.0 kernel, which comes with the newly released CUDA 5.0. We noticed a significant improvement over the old SYMV from CUDA 4.0. However, the improved kernel can be only used by enabling the atomics mode [3]. The second implementation with which we compare ours is the SYMV kernel from CULA [8], a commercial dense linear algebra library for NVIDIA GPUs. Results show that our proposed kernel is still better than all existing implementations.

4 Extending Methodology to GEMV

In this Section, we present the details of the kernel design in the same manner as we did in [7]. First, we begin by the block-level strategy, then we illustrate the thread-level strategy, which we recommend for similar memory-bound kernels.

4.1 Block-Level Strategy

The block-level strategy is simple and straightforward. The input matrix is divided into square blocks. The dimension of the block is a design parameter called `gemv_block_size` and is currently set to 64. This value proved to give the best performance for SYMV [7]. Let us assume for now, that the matrix can be completely divided into 64×64 square blocks. We will show how to manage arbitrary dimensions later on. The kernel is configured to run with `n` thread blocks (TBs), where:

$$\begin{aligned} n = & \text{number_of_rows/gemv_block_size} && \text{for the non-transposed case, or} \\ n = & \text{number_of_columns/gemv_block_size} && \text{for the transposed case.} \end{aligned}$$

As shown in Figure 2, thread blocks originate at the left most blocks in the non-transposed case (Figure 2(a)) and move horizontally from left to right. In the transposed version (Figure 2(b)), they originate at the top blocks and move downwards. The kernel terminates when all TBs finish their horizontal/vertical pass over the matrix.

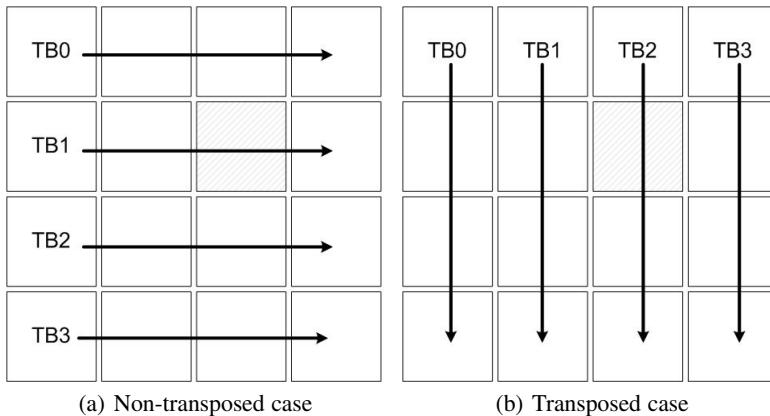


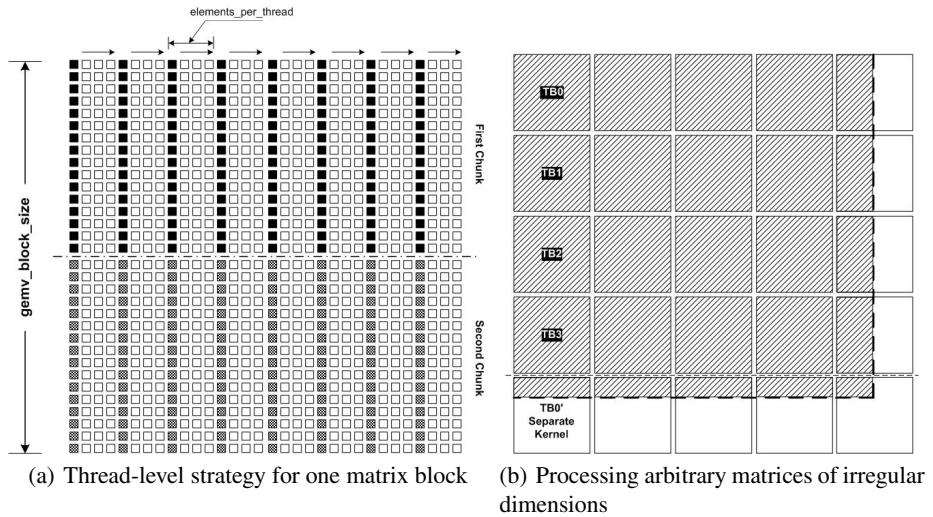
Fig. 2. Movements of thread blocks through the matrix

4.2 Thread-Level Strategy

The thread-level strategy is our main concern. To ease the presentation, we summarize the following design parameters:

- `thread_x` is the x-dimension of each thread block. `thread_x` is always set to `gemv_block_size`.
- `thread_y` is the y-dimension of each thread block.
- `elements_per_thread` is the number of matrix elements a single thread prefetches at one time. `elements_per_thread` is not really a design parameter, and is equal to `gemv_block_size/(2×thread_y)`. However, it puts a restriction of the choice of `thread_y` since it has to be an integer.

Now, let us consider Figure 3(a) as an example. A 32×32 matrix block is processed using 128 threads. Originally, thread blocks are configured as 32×4 threads, meaning that `thread_x=32`, `thread_y=4`, and `elements_per_thread=4`. However, during actual processing, the whole matrix block is subdivided into two chunks, and all threads co-operate in processing each chunk. The reason behind this concept is to introduce more latency hiding for the stalled warps by applying a double buffering scheme. Thread blocks reorganize themselves as 16×8 threads while loading and processing each chunk from memory. Each thread is responsible for processing a number of elements equal to $(2 \times \text{elements_per_threads})$. For the example shown in Figure 3(a), each thread loads four elements from the first chunk. Before processing the first chunk, double buffering comes into play to prefetch the second chunk. Since SMs interleave execution of thread warps to hide the latency of stalled threads, increasing parallelism would lead to more latency hiding. However, we are always constrained by the SM resources of threads and registers (since all loaded elements are prefetched into registers). Similarly, before processing the second chunk, the first chunk of the next matrix block is prefetched. For the non-transposed case, each thread is responsible for the result of two elements in the final resulting vector. In the transposed case, the matrix block is processed in the same manner, but each thread is responsible for the result of `elements_per_thread` elements.

**Fig. 3.** Optimization Techniques

The drawback of this methodology is its intensive register usage, which may lead to low occupancy. Therefore, the tunable parameters introduced above should be carefully chosen depending on the underlying architecture. In particular, on Fermi C2070, thread blocks are configured as 64×8 threads.

One final comment on the proposed methodology is that the final result of an element in the final vector is not owned by one threads. For the non-transposed case, threads having the same `threadIdx.x` share partial results of the same element. For the transposed version, it is `threadIdx.y` instead. In both cases, threads use shared memory to compute the final result through a simple reduction operation.

4.3 Handling Arbitrary Dimension

Consider an arbitrary matrix where rows and columns are not necessarily multiple of `gemv_block_size`. Moreover, the matrix can be rectangular, i.e. rows are not equal to columns. In the presence of clean-up regions, the design is not so simple as in the trivial case shown in Figure 2.

We will consider a non-transposed case shown in Figure 3(b). The other case can be easily understood in a similar way. For the block-level strategy, we separate the processing of the bottom row-of-blocks in an another kernel that is always invoked with one thread block (TB0' in Figure 3(b)). This is because the TB0' will have some branches that are not necessary for TB0 through TB3. And as we recommended in [7], separating different computation strategy may lead to better occupancy. So, we preferred to separate TB0' as it surely consumes more SM resources due to the additional conditionals. To avoid executing the two kernels successively, we benefit from CUDA streams [3] so that the two kernels are concurrently executed. TB0 through TB3 march on from left to right processing blocks without any special treatment until they hit rightmost block,

where some threads positioned after the matrix columns will be inactive. For TB0', it marches on similarly, but takes into account that threads with global row index more than the matrix rows do nothing. When it hits the rightmost block, it encounters the same situation of TB0 through TB3. Inactive threads in an irregular block are programmed to load zero instead of a matrix element. We can say that the matrix is virtually padded with zeros inside registers. The computation takes place similar to a regular block. The engendered extra flops are negligible compared to the overall algorithm complexity.

5 Experimental Results

The results shown in this section were obtained on a single Fermi C2070 GPU card, featuring 448 cores and 6 GB of DRAM. The host machine has a dual-socket quad-core Intel Xeon processor, running at 2.67 GHz, and is equipped with 24 GB of main memory. All kernel implementations were compiled using CUDA 5.0 compiler, except CULA Dense R14, which only supports CUDA driver/runtime version 4.1. The results, for both single precision (SP) and double precision (DP), will be shown in two parts. The first part presents results for the Level 2 BLAS routine GEMV. Both transposed and non-transposed cases are covered. The second part shows the results of two LAPACK routines: The bidiagonal reduction (BRD) and the Hessenberg reduction (HRD). Because it is called at each column/row reduction of the matrix, the GEMV operation represent the main bottleneck. The presented graphs show the impact of improving GEMV on these routines.

5.1 GEMV Performance

Figures 4 and 5 show the performance in Gflop/s for SGEMV and DGEMV, respectively. The improvement achieved in our design comes for relatively small to medium matrix sizes in the non-transposed case, especially for SP. For the non-transposed SP case, the kernels from CUDA 5.0 and CULA Dense R14 are almost identical. Our kernel has an average 80% improvement for matrix sizes below 1700. From 1700 and up to 4300, it drops to 30%. Between 6000 and 12000, our design drops to about 94%

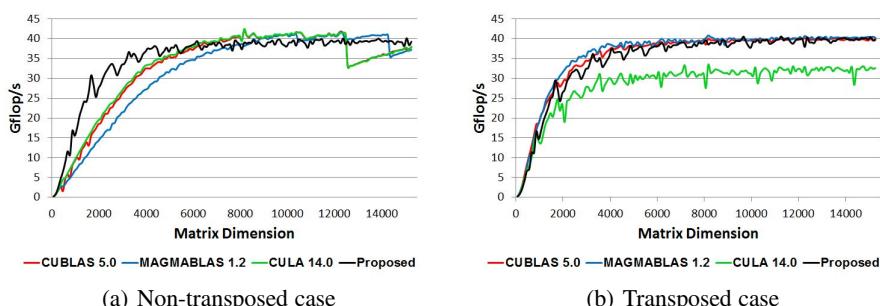


Fig. 4. Performance of GEMV (SP)

of the other designs. A sharp degradation in the performance of the other implementations, starting from 12500, give our kernel an average 10% improvement (from 12500 to 15200).

Doing a similar analysis for double precision, we notice an average 47% improvement up to dimension 1700, then an average 10% improvement between 1700 and 3500. Afterwards, our design has nearly the same performance as the other kernels. As will be shown in Section 5.2, the improvement in the performance for relatively small matrices improves the LAPACK routines for both small and large matrices (again in a relative sense). Our intuition is that the thread-level strategy is able to increase the amount of parallel work even for small matrices. For example, a quick look at the MAGMABLAS GEMV source code informed us that each thread block uses either one 64 or 128 thread vector, compared to 4×64 thread vectors in our case. For large matrices, we probably lose little bit of performance because of the parallel reduction through shared memory.

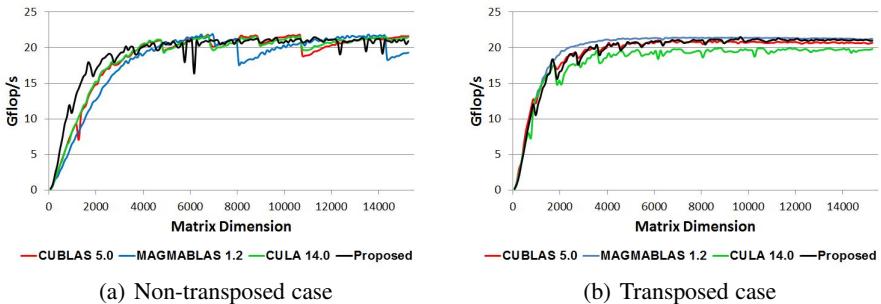


Fig. 5. Performance of GEMV (DP)

5.2 BRD and HRD Performance

In this part, we show the performance for BRD and HRD algorithms, when using the GEMV kernels from CUBLAS 5.0, MAGMABLAS 1.2, and our proposed kernel. The reduction drivers for BRD and HRD are taken from MAGMA. Although CULA has implemented such algorithms, they are not available in the CULA-DENSE free edition.

We see in Figures 6 and 7 that the reduction algorithms perform better when using our GEMV kernel. It beats all other implementations, thanks to our improvements in performance for small matrices. For BRD (SP), the reduction performs 25% better (on average) for dimensions ≤ 8000 , then drops to about 7% improvement. In DP, the average improvement is about 20% for dimensions ≤ 8000 , then drops to 12%. Switching to the HRD algorithm, we achieve an average 35% improvement in SP for dimensions ≤ 8000 , that drops to 7% afterwards. In DP, the improvement is 17% till 6000 then drops to 2%. As we can see, although our GEMV kernel is not competitive for every matrix size, it could lead to a high-level LAPACK driver that is better for every matrix size. The obvious reason is that reduction operations do successive matrix-vector multiplications of decreasing sizes. The conceptual lesson that we learned is this: slight improvements to low-level kernels can not be ignored. There are plenty of higher-level routines where

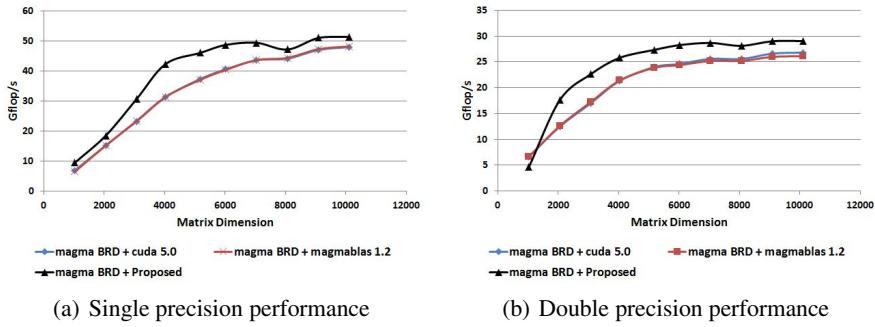


Fig. 6. Performance of bidiagonal reduction using three different GEMV kernels

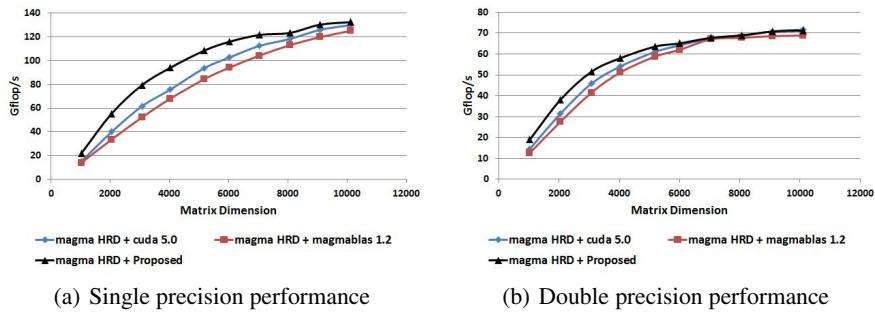


Fig. 7. Performance of Hessenberg reduction using three different GEMV kernels

these kernels lie at their hearts. Such routines can move to another level of performance by slightly enhancing its core kernel.

6 Conclusions and Future Work

This paper introduced an improved implementation of the GEMV kernel on NVIDIA GPU accelerators. The design idea was utilized before in improving the SYMV kernel, which supports our claim that the computation strategy can be regarded as a general approach for processing memory-bound kernels. Although the results shows improvements only for relatively small to medium matrix sizes, the impact on higher-level LAPACK reduction algorithms was significant, even on large matrix sizes.

Possible future plans for this work is to treat the periodic dips in the performance of our GEMV kernel. These dips refer to worst case mapping of TBs on SMs. Since the work load is balanced among all thread blocks, a dip occurs when $(\text{number_of_TBs} \bmod \text{number_of_SMs})$ is relatively small (worst case will be 1). The apparent solution is to involve more workload distribution to maximize the number of active SMs. However, dividing the work of one TB among several SMs requires either reduction operations on the DRAM level. Such trade-off between workload distribution and DRAM access penalty requires further investigation.

References

1. CULA Dense Free Edition, <http://www.culatools.com/>
2. Matrix Algebra on GPU and Multicore Architectures. Innovative Computing Laboratory, University of Tennessee, <http://icl.cs.utk.edu/magma/>
3. NVIDIA CUDA Toolkit, <http://developer.nvidia.com/cuda-toolkit>
4. Nvidia visual profiler, <http://developer.nvidia.com/nvidia-visual-profiler>
5. Performance Application Programming Interface (PAPI). Innovative Computing Laboratory, University of Tennessee, <http://icl.cs.utk.edu/papi/>
6. The NVIDIA CUDA Basic Linear Algebra Subroutines (CUBLAS), <http://developer.nvidia.com/cublas>
7. Abdelfattah, A., Dongarra, J., Keyes, D., Ltaief, H.: Optimizing Memory-Bound SYMV Kernel on GPU Hardware Accelerators. In: The 10th International Meeting on High Performance Computing for Computational Science, VECPAR 2012 (accepted, 2012)
8. Humphrey, J.R., Price, D.K., Spagnoli, K.E., Paolini, A.L., Kelmelis, E.J.: CULA: Hybrid GPU Accelerated Linear Algebra Routines. In: Proceedings of SPIE Defense and Security Symposium, DSS (April 2010)
9. Kurzak, J., Tomov, S., Dongarra, J.: Autotuning GEMM Kernels for the Fermi GPU. IEEE Transactions on Parallel and Distributed Systems PP(99), 1 (2012)
10. Kurzak, J., Luszczek, P., Tomov, S., Dongarra, J.: Preliminary Results of Autotuning GEMM Kernels for the NVIDIA Kepler Architecture - GeForce GTX 680. LAPACK Working Note 267
11. Kwon, Y., Narayanan, R.M., Rangaswamy, M.: A multi-target detector using mutual information for noise radar systems in low snr regimes. In: 2010 International Waveform Diversity and Design Conference, WDD, pp. 000105–000109 (August 2010)
12. Nath, R., Tomov, S., Dong, T., Dongarra, J.: Optimizing symmetric dense matrix-vector multiplication on GPUs. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2011, pp. 6:1–6:10. ACM, New York (2011)
13. Nath, R., Tomov, S., Dongarra, J.: An Improved Magma Gemm for Fermi Graphics Processing Units. Int. J. High Perform. Comput. Appl. 24(4), 511–515 (2010)
14. Volkov, V., Demmel, J.W.: Benchmarking GPUs to tune dense linear algebra. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC 2008, pp. 31:1–31:11. IEEE Press, Piscataway (2008)
15. Yu, W.C., Quan, W.D.: On the signal processing in the life-detection radar using an fmcw waveform. In: 2010 Third International Symposium on Information Processing, ISIP, pp. 213–216 (October 2010)

HiBB 2012: 3rd Workshop on High Performance Bioinformatics and Biomedicine

Mario Cannataro

Bioinformatics Laboratory,
Department of Medical and Surgical Sciences,
University Magna Græcia of Catanzaro,
88100 Catanzaro, Italy
cannataro@unicz.it

Foreword

High-throughput technologies (e.g. microarray and mass spectrometry) and clinical diagnostic tools (e.g. medical imaging) are producing an increasing amount of experimental and clinical data, yielding to the so called age of Big Data in Biosciences. In such a scenario, large scale databases and bioinformatics tools are key tools for organizing and exploring biological and biomedical data with the aim to discover new knowledge in biology and medicine. However the storage, preprocessing and analysis of experimental data is becoming the main bottleneck of the biomedical analysis pipeline.

Parallel computing and high-performance infrastructures are more and more used in all phases of life sciences research, e.g. for storing and preprocessing large experimental data, for the simulation of biological systems, for data exploration and visualization, for data integration, and for knowledge discovery.

The current bioinformatics scenario is characterized by the application of well established techniques, such as parallel computing on multicore architectures and grid computing, as well as by the application of emerging computational models such as graphics processing and cloud computing.

Large scale infrastructures such as grids or clouds are mainly used to store in an efficient manner and to share in an easy way, the huge experimental data produced in life sciences, while parallel computing allows the efficient analysis of huge data. In particular, novel parallel architectures such as GPUs and emerging programming models such as MapReduce, may overcome the limits posed by conventional computers to the analysis of large amounts of data.

The third edition of the Workshop on *High Performance Bioinformatics and Biomedicine* (HiBB) aimed to bring together scientists in the fields of high performance computing, computational biology and medicine to discuss the parallel implementation of bioinformatics algorithms, the deployment of biomedical applications on high performance infrastructures, and the organization of large scale databases in biology and medicine. As in the past, also this year the workshop has been organized in conjunction with Euro-Par, the main European (but international) conference on all aspects of parallel processing.

The workshop included the invited talk *Using Clouds for Scalable Knowledge Discovery Applications* that was held by Prof. Domenico Talia (University of Calabria and ICAR-CNR, Italy), and presentations were organized in three sessions: *High Performance Bioinformatics and Systems Biology*, *Software Platforms for Bioinformatics and Biomedicine*, and *Grid and Cloud Computing for Life Sciences*.

The invited talk *Using Clouds for Scalable Knowledge Discovery Applications* discussed the use of clouds for the development of scalable distributed knowledge discovery applications for life sciences, and presented a software framework for implementing high-performance data mining applications modelled through workflows on clouds.

The first session, *High Performance Bioinformatics and Systems Biology*, comprised three papers discussing the parallel implementation of bioinformatics and systems biology algorithms.

In the first paper, *P3S: Protein Structure Similarity Search*, Galgonek et al. presented the P3S web-based tool for similarity search in protein structure databases, that uses an efficient indexing technique and the SProt specialized similarity measure.

A second paper by Galgonek et al., *On the Parallelization of the SProt Measure and the TM-score Algorithm*, discussed the parallel implementation of the SProt similarity measure.

In the third paper, *Stochastic Simulation of the Coagulation Cascade: A Petri Net Based Approach*, Castaldi et al. presented a Stochastic Petri Net (SPN) based model used to introduce uncertainty to capture the variability of biological systems. The authors described the SPN model for the Tissue Factor induced coagulation cascade and simulated it using Gillespie's Tau-Leaping Stochastic Simulation, proving that a stochastic approach allows to detect important features that deterministic models cannot discover.

The second session, *Software Platforms for Bioinformatics and Biomedicine*, comprised three papers describing experiences on using Grid and Cloud computing as well as multicore and GPUs architectures, to implement biomedical data analysis.

In the first paper, *Processing the biomedical data on the grid using the UNICORE workflow system*, Borcz et al. described the use of the UNICORE middleware to automate the analysis pipeline in biomedical applications, with focus on the determination of the spectrum of mutations in mitochondrial genomes of normal and colorectal cancer cells, using the PL-Grid Polish National Grid Infrastructure.

In the second paper, *Multicore and Cloud-based Solutions for Genomic Variant Analysis*, Gonzalez et al. introduced the High Performance Genomics (HPG) project that is developing a collection of open-source tools for genomics data, and presented the HPG Variant, a suite for genomic variant analysis that allows to conduct genomic-wide and family-based analysis.

In the third paper, *A Novel Implementation of Double Precision and Real Valued ICA Algorithm for Bioinformatics Applications on GPUs*, Foshati et al.

presented the parallel implementation on a GPU using the CUDA programming toolkit, of an ICA (Independent Component Analysis) algorithm named Joint Approximate Diagonalization of Eigen-matrices (JADE), for the analysis of EEG signals.

The third session, *Grid and Cloud Computing for Life Sciences*, included the aforementioned invited talk, a paper describing an environment for scientific workflows on heterogeneous grids, and a panel on *High Performance Bioinformatics and Biomedicine*.

In particular, in the paper *PROGENIA: an approach for Grid Interoperability at Workflow level*, Mirto et al. presented the PROGENIA Workflow Management System that supports interoperability at workflow level. The presented system allows the deployment of scientific workflows on different grids based on Globus, gLite, and Unicore middlewares, and has been tested on several bioinformatics case studies.

The workshop was concluded by a panel on *High Performance Bioinformatics and Biomedicine* chaired by Mario Cannataro. The goal of the panel was to discuss emerging topics and problems regarding the application of High Performance Computing techniques, such as Grid and Cloud Computing, to biomedical, bioinformatics, and systems biology applications.

The three panelists, Francesco Archetti, Piotr Bala, and Domenico Talia, were asked to answer the following questions:

- The role of systems biology to speed up the development of new drugs.
- Role of grids and clouds in high performance bioinformatics and biomedicine.
- Integration and analysis of omics and clinical data.

This post-workshop proceedings includes the final revised versions of the HiBB papers and invited talk, taking the feedback from reviewers and workshop audience into account.

The program chair sincerely thanks the Program Committee members and the following additional reviewers: Giuseppe Agapito, Marco Aldinucci, Barbara Calabrese, Maria Mirto, and Simona Rombo, for the time and expertise they put into the reviewing work, the Euro-Par organization, for providing the opportunity to arrange the HiBB workshop in conjunction with the Euro-Par 2012 conference, and all the workshop attendees who contributed to a lively day.

October 2012

Mario Cannataro

Using Clouds for Scalable Knowledge Discovery Applications

Fabrizio Marozzo¹, Domenico Talia^{1,2}, and Paolo Trunfio¹

¹ DIMES, University of Calabria

² ICAR-CNR

Via P. Bucci, Cubo 41c

87036 Rende(CS), Italy

{fmarozzo,talia,trunfio}@deis.unical.it

Abstract. Cloud platforms provide scalable processing and data storage and access services that can be exploited for implementing high-performance knowledge discovery systems and applications. This paper discusses the use of Clouds for the development of scalable distributed knowledge discovery applications. Service-oriented knowledge discovery concepts are introduced, and a framework for supporting high-performance data mining applications on Clouds is presented. The system architecture, its implementation, and current work aimed at supporting the design and execution of knowledge discovery applications modeled as workflows are described.

1 Introduction

Cloud platforms, HPC systems, and large-scale distributed computing systems can be used to solve big and complex problems in several scientific and business domains. In particular, the huge amount of data available today in digital repositories requires smart data analysis techniques and scalable algorithms, techniques, and systems to help people to deal with it.

Cloud computing is a paradigm and a technology that provide location-independent storing, processing and use of data on remote computers accessed over the Internet. Users can exploit almost unlimited computing power on demand, thus they do not need to buy hardware and software to fulfill their needs [1].

Users can employ Cloud systems to store any kind of information and to use software as a service (e.g., office automation tools, music players, games). Organizations, in science, business and public services, can use Cloud services to run data centers and implement different kinds of IT activities. People and enterprises can leverage Clouds to scale up their activities without investing in large physical infrastructures.

Data mining techniques are widely used in several application areas for analyzing and extracting useful knowledge from large datasets. In most cases, the core of data analysis applications is represented by compute-intensive data mining algorithms, thus leading to very long execution times when a single computer

is used to analyze a large dataset. Cloud systems can be effectively used to handle data mining processes since they provide scalable processing and storage services, together with software platforms for developing knowledge discovery systems on top of such services.

For instance, scalable computing systems give a fundamental support to life science discoveries in several aspects:

- Distributed/parallel data management and processing,
- Scalable data integration and interoperability,
- Distributed/parallel data mining,
- Collaborative data exploration and visualization.

Considering this scenario, we worked to design a framework for supporting the scalable execution of knowledge discovery applications on top of Cloud platforms. The framework has been designed to be implemented on different Cloud systems; however, an implementation of this framework has been carried out using Windows Azure¹ and has been evaluated through a set of data analysis applications executed on a Microsoft Cloud data center.

The framework has been designed to support three classes of knowledge discovery applications: *single-task applications*, in which a single data mining task such as classification, clustering, or association rules discovery is performed on a given dataset; *parameter-sweeping applications*, in which a dataset is analyzed by multiple instances of the same data mining algorithm with different parameters; *workflow-based applications*, in which knowledge discovery applications are specified as graphs linking together data sources, data mining tools, and data mining models.

The remainder of this paper is structured as follows. Section 2 discusses the system architecture, execution mechanisms, and an early version of the user interface designed to support single-task and parameter-sweeping applications. Section 3 describes current work aimed at providing a Web-based interface and associated execution mechanisms for designing and running knowledge discovery applications as workflows. Finally, Section 4 concludes the paper.

2 System Architecture and Implementation

The architecture of the designed framework, shown in Figure 1, includes the following components:

- A set of binary and text data containers used to store data to be mined (*input datasets*) and the results of data mining tasks (*data mining models*).
- A *Task Queue* that contains the data mining tasks to be executed.
- A *Task Status Table* that keeps information about the status of all tasks.
- A pool of k *Workers*, where k is the number of virtual servers available, in charge of executing the data mining tasks resulting from the data mining applications submitted by the users.

¹ <http://www.microsoft.com/windowsazure>

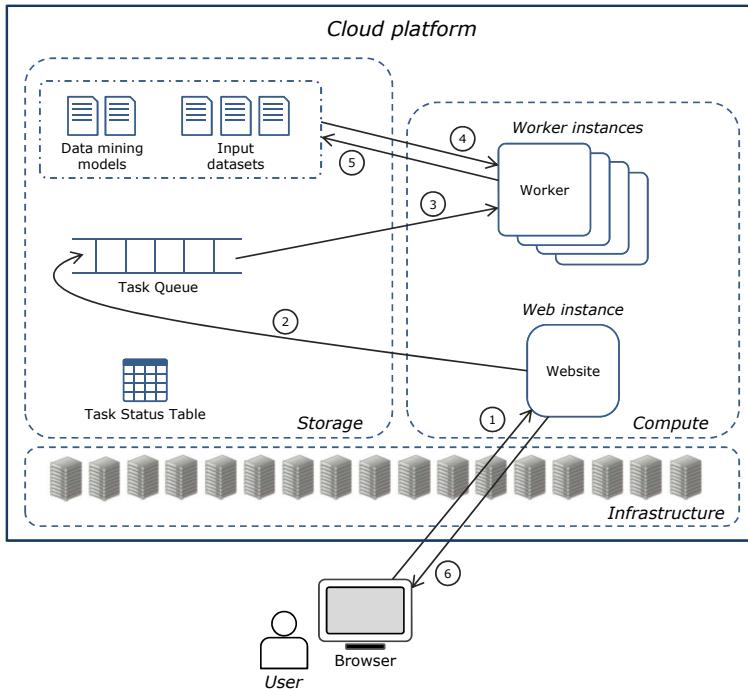


Fig. 1. System architecture and application execution steps

- A *Website* that allows users to submit, monitor the execution, and access the results of their knowledge discovery applications.

2.1 Applications Execution

The following steps are performed to develop and execute a knowledge discovery application through the system (see Figure 1):

1. A user accesses the Website and develops her/his application (either single-task, parameter-sweeping, or workflow-based) through a Web-based interface. After completing the application, she/he can submit it for execution on Azure.
2. After the application submission, a set of tasks are created and inserted into the Task Queue on the basis of the application submitted by the user.
3. Each idle Worker picks a task from the Task Queue, and starts its execution on a virtual server.
4. Each Worker gets the input dataset from the location specified by the application. To this end, a file transfer is performed from the container where the dataset is located, to the local storage of the virtual server the Worker is running on.

5. After task completion, each Worker puts the result on a data storage element.
6. The Website notifies the user as soon as her/his task(s) have completed, and allows her/him to access the results.

The set of tasks created on the second step depends on the type of application submitted by the user. In the case of a single-task application, just one data mining task is inserted into the Task Queue. If the user submits a parameter-sweeping application, one task for each combination of the input parameters values is executed. In the case of a workflow-based application, the set of tasks created depends on how many data mining tools are invoked within the workflow. The Task Status Table is dynamically updated whenever the status of a task changes. The Website periodically reads and shows the content of such table, thus allowing users to monitor the status of their tasks.

Input data are temporarily staged on a server for local processing. To reduce the impact of data transfer on the overall execution time, it is important that input data are physically close to the virtual servers where the workers run on. For example, in the Azure implementation of our framework, this had been done by exploiting the Azure's *Affinity Group* feature, which allows storage and servers to be located near to each other in the same data center for optimal performance.

Currently, the framework includes a wide range of data mining algorithms from Weka [2], and supports the *arff* format for the input datasets. Since the Weka algorithms are written in Java, each Worker includes a Java Virtual Machine to run the corresponding data mining tasks.

2.2 User Interface

The user interface of the system is composed of two main parts: one pane for composing and running both single-task and parameter-sweeping applications and another pane for composition and execution of workflow-based knowledge discovery applications.

The first part of the user interface (the Website), presented in [3], was implemented to support single-task and parameter-sweeping applications. It includes three main sections: i) *Task submission* that allows users to submit their applications; ii) *Task status* that is used to monitor the status of tasks and to visualize results; iii) *Data management* that allows users to manage input data and past results.

Figure 2 shows a screenshot of the Task submission section, taken during the execution of a parameter-sweeping application. An application can be configured by selecting the algorithm to be executed, the dataset to be analyzed, and the relevant parameters for the algorithm. The system submits to the Cloud a number of independent tasks that are executed concurrently on a set of virtual servers.

Fig. 2. Screenshot of the Task submission section

The user can monitor the status of each single task through the Task status section, as shown in Figure 3. For each task, the current status (submitted, running, done or failed) and status update time are shown. Moreover, for each task that has completed its execution, two links are enabled: the first one (*Stat*) gives access to a file containing some statistics about the amount of resources consumed by the task; the second one (*Result*) visualizes the task result.

3 Cloud Knowledge Discovery Workflows

We prototyped the programming interface and its services to support the composition and execution of workflow-based knowledge discovery applications in our Cloud framework. Workflows support research and scientific processes by providing a paradigm that may encompass all the steps of discovery based on the execution of complex algorithms and the access and analysis of scientific data. In data-driven discovery processes, knowledge discovery workflows can produce results that can confirm real experiments or provide insights that cannot be achieved in laboratories.

Following the approach proposed in [5] and [6], we model a knowledge discovery workflow as a graph whose nodes represent resources (datasets, data mining tools, data mining models), implemented as Cloud services, and whose edges represent dependencies between resources.

To support the workflow composition, we implemented the Website part that, by using native HTML 5 features, allows users to design service-oriented knowledge discovery workflows with a simple drag-and-drop approach. Figure 4 shows a screenshot of the Website taken during the composition of a knowledge discovery workflow.

Home	Task submission	Task status	Data management	About	
TASK STATUS					
Task ID	CurrentStatus	StatusUpdateTime	Statistics	Result	Archive
1634454118824362358-001	done	7/4/2011 7:34:08 PM	Stat	Result	
1634454118824362358-002	done	7/4/2011 7:33:10 PM	Stat	Result	
1634454118824362358-003	done	7/4/2011 7:34:00 PM	Stat	Result	
1634454118824362358-004	done	7/4/2011 7:34:19 PM	Stat	Result	
1634454118824362358-005	running	7/4/2011 7:33:11 PM	Stat	Result	
1634454118824362358-006	running	7/4/2011 7:34:00 PM	Stat	Result	
1634454118824362358-007	running	7/4/2011 7:34:09 PM	Stat	Result	
1634454118824362358-008	running	7/4/2011 7:34:20 PM	Stat	Result	
1634454118824362358-009	submitted	7/4/2011 7:32:14 PM	Stat	Result	
1634454118824362358-010	submitted	7/4/2011 7:32:15 PM	Stat	Result	

Fig. 3. Screenshot of the Task status section

On the top-left of the window, three icons allow a user to insert a new dataset, tool, or model node into the workflow. Once placed into the workflow, a node can be linked to others to establish the desired dependencies. Relevant information for each node (e.g., the location for a dataset, algorithm name and associated parameters for a tool) can be specified through a configuration panel available on the right of the window.

As an example, the workflow shown in Figure 4 implements a knowledge discovery process known as *bagging*. The workflow begins, on the left, with the input dataset (*CoverType*) connected to a Splitter tool that extracts four samples from it. The first three samples are used as training sets by three instances of the J48 classification tool (an open source implementation of C4.5 [4]), which generate three independent classification models from them. Then, a Voter tool receives the three models and the fourth sample as test set, and produces the final classification model through a voting procedure.

The five tools invoked in the workflow (the Spitter, three J48 instances, and the Voter) are translated into an equal number of tasks, indicated as $T_1 \dots T_5$ in Figure 4. Differently from parameter-sweeping applications whose tasks are independent each other and therefore can be executed in parallel, the execution order of workflow tasks depends on the dependencies specified by the workflow edges. To ensure the correct execution order, each workflow task is associated with a list of tasks that must be completed before starting its execution. For instance, Figure 5 shows the content of the Task Queue after having submitted the workflow shown in Figure 4. For each task, the list of tasks to be completed before its execution is reported. According with the tasks dependencies specified by the workflow, after completion of T_1 , the execution of T_2 , T_3 and T_4 can proceed concurrently. Moreover, T_5 can be executed only after completion of T_2 , T_3 and T_4 .

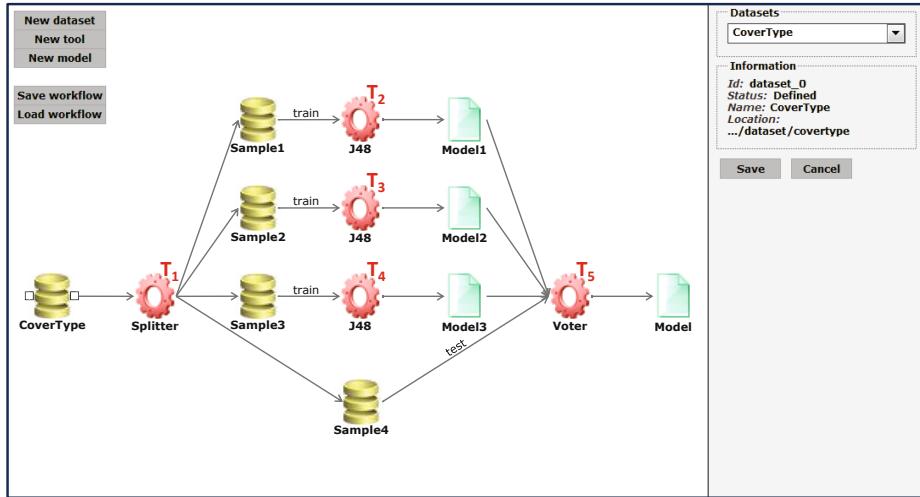


Fig. 4. Workflow composition interface

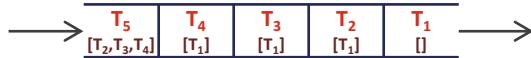


Fig. 5. Content of the Task Queue after submission of the workflow in Figure 4

4 Conclusions

We need new distributed infrastructures and smart scalable analysis techniques to solve more challenging problems in science. Cloud computing systems can be effectively used as scalable infrastructures for service-oriented knowledge discovery applications. Based on this vision, we designed a Cloud-based framework for large-scale data analysis.

We evaluated the performance of the system through the execution of a set of long-running parameter-sweeping knowledge discovery applications on a pool of virtual servers hosted by a Microsoft Cloud data center. The experimental results, presented in [3], demonstrated the effectiveness of the framework, as well as the scalability that can be achieved through the execution of parameter-sweeping applications on a pool of virtual servers. For example, the classification of a large dataset (290,000 records) on a single virtual server required more than 41 hours, whereas it was completed in less than 3 hours on 16 virtual servers. This corresponds to an execution speedup equal to 14.

Currently, we are working on the workflow composition interface with the aim of extending the supported design patterns (e.g. conditional branches and iterations) and to experimentally evaluate its functionality and performance on Windows Azure during the design and execution of complex knowledge discovery workflows on large data on the Cloud.

References

1. The European Commission. Unleashing the Potential of Cloud Computing in Europe. Brussels (2012)
2. Witten, H., Frank, E.: Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann Publishers (2000)
3. Marozzo, F., Talia, D., Trunfio, P.: A Cloud Framework for Parameter Sweeping Data Mining Applications. In: Proc. of the 3rd International Conference on Cloud Computing Technology and Science, CloudCom 2011, Athens, Greece, pp. 367–374 (2011)
4. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers (1993)
5. Cesario, E., Lackovic, M., Talia, D., Trunfio, P.: A Visual Environment for Designing and Running Data Mining Workflows in the Knowledge Grid. In: Holmes, D.E., Jain, L.C. (eds.) Data Mining: Foundations and Intelligent Paradigms. ISRL, vol. 24, pp. 57–75. Springer, Heidelberg (2012)
6. Talia, D., Trunfio, P.: Service-Oriented Distributed Knowledge Discovery. Chapman and Hall/CRC Press (2012)

P3S: Protein Structure Similarity Search

Jakub Galgonek, Tomáš Skopal, and David Hoksza

Departement of Software Engineering, Charles University in Prague
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
{galgonek,skopal,hoksza}@ksi.mff.cuni.cz

Abstract. Similarity search in protein structure databases is an important task of computational biology. To reduce the time required to search for similar structures, indexing techniques are being often introduced. However, as the indexing phase is computationally very expensive, it becomes useful only when a large number of searches are expected (so that the expensive indexing cost is amortized by cheaper search cost). This is a typical situation for a public similarity search service. In this article we introduce the P3S web application (<http://siret.cz/p3s>) allowing, given a query structure, to identify the set of the most similar structures in a database. The result set can be browsed interactively, including visual inspection of the structure superposition, or it can be downloaded as a zip archive. P3S employs the SProt similarity measure and an indexing technique based on the LAESA method, both introduced recently by our group. Together with the measure and the index, the method presents an effective and efficient tool for querying protein structure databases.

Keywords: protein structure, similarity retrieval, web service.

1 Introduction

Proteins have a wide range of functions in living organisms such as enzymatic, signaling, transportation and building function. The way proteins carry out their biological function is through interaction with other proteins or small molecules based on spatial arrangement of their physicochemical properties. Therefore, identification of similar tertiary folds can bring invaluable insight into function of proteins with known 3D structures [1]. The similarity methods can be divided into two groups — global similarity measures taking into account the whole structure, and local similarity measures comparing only active sites, that is, the sites where the protein links to its binding ligand or protein. The latter methods can be more precise in linking functional similarity to structural similarity since they focus directly on the site where the action occurs. On the other hand, for new structures the active site is often not known as localizing the exact ligand position requires the structures to be resolved at high resolution. In the current version (v.2011) of sc-PDB [2], the annotated database of druggable binding sites from the Protein DataBank (PDB) [3] contains binding site information for 3034 proteins and, at the same time (March 27, 2012), PDB contains 3D information for 80,402 structures. This disproportion favors the first group of

methods which are able to assess pairwise protein structure similarity without the need of knowing exact binding site positions. There have been many methods proposed for assessing protein similarity on the tertiary structure level in recent years [4–9]. However, one can see the trend of addressing not only the effectiveness (quality) of the comparison process but also its efficiency (speed). The shift can be clearly attributed to the growth of the number of structures in PDB. On the other hand, we stress out that the need for development of fast methods is still not saturated if we consider the difference in the number of structures deposited in sequence and structure databases. Currently (March 27, 2012), PDB contains 80,402 structures in comparison to 535,248 sequences deposited in UniProtKB/Swiss-Prot [1]. Since it has been shown that the structure is more conserved than sequence [10] there is a strong demand on increasing the quality and speed of structure determination techniques. In turn, the demand results in more structures with the ultimate goal to fully cover the sequence space by structural analogues.

Recently, we have contributed to the global protein structure similarity area by introducing a method called SProt [11]. SProt not only focuses on high-quality structure alignment but also introduces database indexing techniques applicable to the proposed measure, thus greatly improves the efficiency when used for searching large protein structure databases.

To further address the needs of bioinformatics community, the methods should provide not only similarity search of user-uploaded structures in protein databases but also subsequent visual inspection of the superposition. That can further help in explanation of the functional relation between the query and the results. Although there exist some exceptions (see section 3.1), these requirements are the weak points of many other methods. In this article, we present a freely available interactive web application for similarity search in protein structure databases allowing querying a database of protein structures using a single query structure. It utilizes the previously presented SProt [11] method which proved to be a highly effective and efficient addition to the portfolio of protein structure similarity methods. The application allows to visually inspect the alignments and the superpositions of the results with the query structure, while it also provides an export of the results for later use.

2 Implementation

In the following section we briefly summarize the technologies that have been used to implement our system. Subsequently, we focus on the description of the usage of the web application.

2.1 Web Server

From the beginning, the P3S web server has been considered as an interactive web application. Because of this intent, we use Google Web Toolkit¹ that has

¹ <https://developers.google.com/web-toolkit>

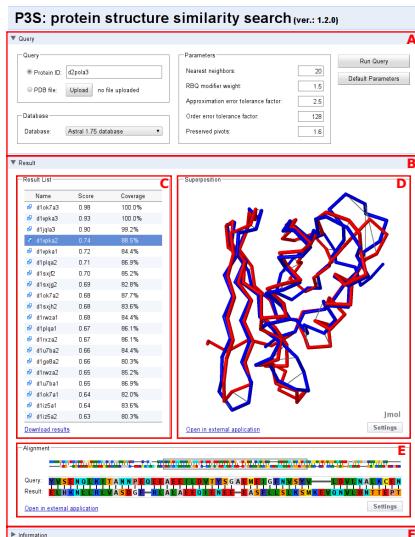


Fig. 1. The P3S web server user interface

been designed especially for writing such type of applications. The framework allows to develop the client-side part of the application (running in a user's web browser) together with the required application server. The Jmol java applet² has been used for the dynamic visualization of the proteins.

The computational server employs the SProt similarity measure and an indexing technique based on the LAESA method (adjusted specifically for the SProt measure), both introduced recently by our group [11]. In order to obtain a high-performance solution, the computational server has been developed in C++ using *Intel Threading Building Blocks* (TBB) library [12] that provides parallel implementation. The computational server is running on a dedicated server and communicates with the application server using the well-established CORBA technology.

2.2 Usage of P3S

The web server is available at the address <http://siret.cz/p3s>. The user interface is divided into three panels — Query (Fig. 1A), Result (Fig. 1B) and Information (Fig. 1F). The Query panel is used to submit a query. The Result panel is used to present the retrieved structures and the Information panel shows link to the application documentation and other relevant information.

Query Submission. To perform a search, the user has to select a query structure, the target database, and the number of the most similar structures which are to be retrieved from the database.

² <http://www.jmol.org>

The query structure can be defined by its ID or uploaded manually. In the first case the structure stored on the server having the given code is used. The second choice is to upload user-defined PDB file. The PDB file should use the actual version of the PDB format and should also include all heavy atoms. The application allows checking the messages from the PDB parser, which is useful in cases when the PDB file is rejected due to errors in the format. If the PDB file contains multiple models, only the first one is taken into account.

Currently, P3S supports two databases that the user can choose. The first one is the Astral [13] 1.75 database containing 33,352 structures (only structures having different sequences are included). The second one is the ProtDex2 database containing 34,055 structures [9]. It is applicable mainly for comparison with other methods. Note that the databases do not contain multi-domain protein structures. Therefore the query should be preferably a single-domain structure.

The number of the most similar structures to be retrieved can be set together with other parameters that influence the behavior of the access method. The most important parameter is so-called *RBQ modifier weight*, which controls the efficiency vs. precision tradeoff.

Although setting the number of retrieved structures and the RBQ modifier weight can speedup the search substantially, the time needed for the evaluation of a query depends also on the size of the query structure and on the number of (similar) structures in the database. To give the user information on the query progress, the query progress bar shows the fraction of the already searched database. Note that because of the non-linear behavior of the search algorithm the progress pace may not be constant during the querying.

Result Presentation. When the query process completes, the retrieved structures are presented in the result list (Fig. 1C) containing several columns. The *Name* column describes the SCOP ID [14] of the result structure. The *Score* column shows similarity score between the query structure and given result structure. The score range goes from 0 to 1. The last column, *Coverage*, determines the coverage of the query structure, i.e., percentage of the amino acids of the query that were aligned with some amino acid of the given result structure. The leftmost column contains links to the PDB server, showing detailed information about the selected structure. Structures are sorted according to the *Score* column.

When selecting an item from the result list, visualizations of the superposition (Fig. 1D) and the alignment (Fig. 1E) between the query structure and the selected structure are shown. The orientation of the query structure is not changed when switching between the items. Hence, if the user focuses on some part of the query structure, switching to other result item does not distract her/him.

The alignment and superposition visualizations are interlinked so that one can inspect the alignment on both sequence and structure level. In the superposition visualization, the lines between the C_{α} atom positions indicate the aligned residues. Furthermore, when the mouse hovers over an amino acid in the alignment visualization, the corresponding amino acid is highlighted in the

superposition visualization and vice versa. Both of the visualizations can be customized using the *Settings* menu placed in the lower right corner.

Results Download. The results can be also exported in the form of a zip archive which allows a possible later analysis of the obtained results. The archive includes the query PDB file, a text file describing the query parameters and a text file containing the brief descriptions (name, score and coverage) of the obtained result set. For each retrieved structure, the zip archive contains subfolder with other useful information related to the retrieved structure — the original PDB file, the superimposed PDB file, the superposition (in the Jmol format), the alignment (in the FASTA format), and the information file containing additional information about the retrieved structure, especially the transformation defining the superposition.

If the user does not want to use the Jmol for the visualization/analysis of the superposition, (s)he has two possibilities. The user can just enter the PDB file with query structure and the PDB file with superimposed result structure into a program of her/his choice. Another possibility is to use the original PDB file and apply the transformation stored in the information file.

3 Results

In this section, we focus on a comparison of our web application with other current web applications used for similarity search in protein structure databases. To evaluate the effectiveness of P3S we compare it to other, not necessarily web-based, methods.

3.1 Comparison of Web Application Interfaces

For our first comparison, we have selected applications which we think represent the current state of the art in the field, namely: Vorometric³ [5], ProBiS⁴ [15], deconSTRUCT⁵ [16], PDBeFold⁶ [17], VAST⁷ [18], iSARST⁸ [19], 3D-BLAST⁹ [20], and Dali server¹⁰ [21]. The summary of basic features of the application is shown in Table 1.

Each of the compared web applications allows to select a query structure by its SCOP ID (see row *support SCOP ID*) or by its PDB ID (see row *support PDB ID*). However, much more important is the ability to upload a user’s protein structure. This feature is supported by all the applications except Vorometric

³ <http://bio.cse.ohio-state.edu/Vorometric/>

⁴ <http://probis.cmm.ki.si/>

⁵ http://epsf.bmad.bii.a-star.edu.sg/struct_server.html

⁶ <http://www.ebi.ac.uk/msd-srv/ssm>

⁷ <http://www.ncbi.nlm.nih.gov/Structure/VAST/>

⁸ <http://sarst.life.nthu.edu.tw/iSARST/>

⁹ <http://3d-blast.life.nctu.edu.tw/>

¹⁰ http://ekhidna.biocenter.helsinki.fi/dali_server/

Table 1. Basic features of web applications

	P3S	Vorometric	ProBiS	deconSTRUCT	PDBeFold	VAST	iSARST	3D-BLAST	Dali server
upload PDB file	✓	—	✓	✓	✓	✓	✓	✓	✓
support SCOP ID	✓	✓	—	—	✓	—	✓	✓	—
support PDB ID	✓	✓	✓	✓	✓	✓	✓	✓	✓
email notification	—	✓	✓	✓	—	—	—	—	✓
search history	—	✓	—	—	—	—	✓	—	—
presentation scheme	C	A	C	A	A	B	A	A	B
external links	✓	✓	✓	✓	✓	✓	✓	✓	—
visual inspection	✓	✓	✓	✓	✓	— ¹	✓	✓	✓
superposition download	✓	✓	✓	✓	✓	✓	✓	✓	✓
alignment download	✓	✓	—	✓	✓	—	—	—	—
full download (zip)	✓	✓	— ²	✓	— ²	— ²	—	— ²	— ²

¹ Superpositions can be visualized by an external application.

² It is possible to download a text-based result list.

(see row *upload PDB file*). Some of the applications also allow to specify an email address to which a notification will be sent when the search is finished (see row *email notification*). It is very useful mainly for applications where searching takes a very long time.

Two of the applications also support searching history (see row *search history*) that allows to show results of older queries. In Vorometric, the history is showed after entering an email address. We think that it is not an optimal solution as the search history of a person can be displayed to anyone who knows the email address. The history of iSARST is based on the web browser session, so its usage is limited.

In the scheme of the presentation of retrieved structures, we can observe three basic types (see row *presentation scheme*). The result of searching is often presented as a list of the retrieved structures, while a separate page (type A) or separate pages (type B) are used to display the result details. The separate pages allow to display more details, however, the browsing through retrieved structures is not so comfortable. Instead, P3S (and ProBiS) display the result list together with the details about a selected structure (type C) on a single page.

The most important part of each application is the presentation of results itself — it includes the visualization of the alignment and the visualization of the superposition between a query structure and a retrieved structure. Except VAST that uses an external application, all applications display superpositions as an integral part of the web application (see row *visual inspection*). For this purpose, the Jmol java applet is used. Some of the applications use the superposition visualization also to show other details, e.g., to highlight aligned parts. However, the connection between the alignment visualization and the superposition visualization is a unique feature of P3S. Except Dali server, the other applications also present a link to other external resources (for example into PDB or SCOP database).

Another important part of the applications is the possibility to download the results. All applications allow to download the superposition of a query and a selected result structure (see row *superposition download*). On the other hand, the options to download the structure-based alignment for the select result structure are quite limited (see row *alignment download*). Also a possibility to download all results as a single file archive is not often supported, although many applications have the possibility to download text files with the list of the retrieved structures (see row *full download (zip)*).

Based on the above comparison we consider the interface of P3S to be comparable with interfaces of other web applications.

3.2 Comparison of Databases

Next, we focus on the supported databases and the query times. The results are summarized in Table 2.

To obtain a result in a reasonable time, large databases are generally employed by applications that use fast algorithms and simpler models. They are also employed by applications that do not present the result instantly. The applications based on slower algorithms using more precise models employ smaller but representative structure databases. P3S belongs to the second category. The supported databases are present in column *Database*, the size of the largest supported databases is presented in column *Max. size*.

Due to the differences in the databases, due to the hardware and also due to differences in weakness points of the respective algorithms, it is problematic to compare a query time of the applications. The query time is present in column *Query time* and represents a time declared by the respective authors wherever possible. In other cases, we present an approximate time based on our observation. Although these times cannot be used for rigorous comparison, they reflect the user experience time very well.

Although P3S uses an algorithm based on a precise model, the achieved query times are reasonably low to be considered as instant.

3.3 Effectiveness of P3S

Qualities of the SProt measure and its indexing method have been evaluated in the original paper describing the method [11]. In this new article, we want to introduce a modified experiment that evaluates the overall quality of the measure and its indexing method.

For this purpose we adopted the experimental setup from the area of *information retrieval*. Such a setup comprises of a database of objects that a tested system uses for retrieval and a query set of objects. For each object from the query set, only a (small) subset of the database is considered to be relevant by the author of the experiment. When the information system returns the result for the given query, the quality of such result can be described in terms of *precision* and *recall*. Precision expresses how many percent of objects in the result set are relevant. Recall expresses how many percent of all relevant objects are obtained

Table 2. Used databases and query times

	Database	Max. size	Query time
P3S	SCOP/ASTRAL-100 (ver. 1.75), ProtDex2 dataset (SCOP/ASTRAL)	34,055	The search takes seconds to minutes.
Vorometric ProBiS	SCOP/ASTRAL-25 (ver. 1.73) PDB (non-redundant structures)	6,981 30,300	The search may take a few minutes. Computation for a medium sized protein will take around 30-60min.
deconSTRUCT	PDB chains + representative subsets	179,543	Based on our observation, the search takes seconds to minutes.
PDBeFold	PDB chains, SCOP/ASTRAL (ver. 1.73) + subset	192,994	Most requests are completed within 1 or 2 minutes.
VAST	PDB + medium-redundancy subset	N/A	Based on our observation, the search takes minutes to tens of minutes.
iSARST	PDB chains (May 2011) + subsets, SCOP/ASTRAL (ver. 1.73) + subsets	176,690	A typical search along with superimposing 100 structures takes only 3~5 seconds.
3D-BLAST	PDB chains (03-Mar-12) + nr subset, SCOP/ASTRAL (ver. 1.75) + subsets	173,232	The method search more than 10000 protein structures in 1.3 seconds.
Dali server	PDB chains	190,642	Most queries finish within an hour.

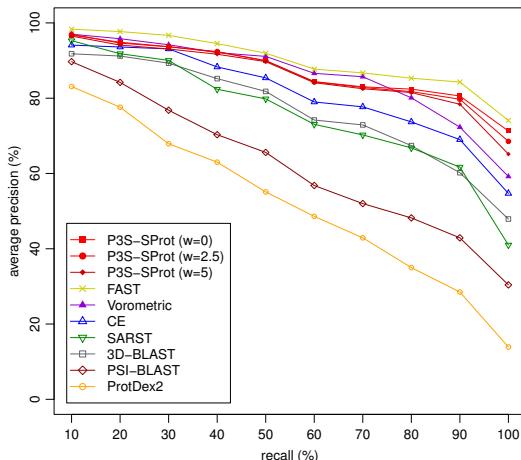
in the result set. These two qualities usually tend to be in a contradiction. An effort that increases the precision decreases the recall, and vice versa.

To evaluate the dependency between precision and recall for an average query, the whole query set instead of just one query is used. For each of the query from the query set, the precision at the given recall level is computed and averaged across the whole query set. This is performed for different values of recall plotted in the precision-recall graph.

In our experiment, we used the ProtDex2 dataset consisting of 34,055 proteins. As the query set, 108 structures from medium-size families of the dataset were selected [9]. The retrieved structure is considered to be relevant to the query if it comes from the same SCOP family [14]. This dataset is widely used, so a comparison with other methods can be performed. For the comparison, we have selected the following methods: FAST [4], Vorometric [5], CE [6], SARST [7], 3D-BLAST, [8], PSI-BLAST [22], and ProtDex2 [9]. The graph data for the compared methods are borrowed from [5] and [7].

We tested P3S for different values of the weight of the RBQ modifier, while other parameters remained unchanged. The reason for such decision was the weight had the greatest impact on the effectiveness and efficiency, because it was the only parameter directly affecting the measure and its indexability.

Figure 2 shows the results of the experiment. With increasing weight w , the precision of the method is decreasing, especially for high recall levels. Because for $w > 2.0$ the impact on speed is negligible, we strongly recommend using the weight in the range $\langle 0, 2.5 \rangle$. As the figure shows, P3S has better precision-recall curve than the other methods, except Vorometric and FAST. In comparison with Vorometric, the curves of P3S are the same or slightly worse for medium recall levels, while they are noticeably better for high recall levels. FAST slightly outperforms P3S but on the other hand it does not offer a web search server as P3S does and therefore is not easily accessible to wide scientific community which makes P3S one of the most precise publicly available web application in the dataset.

**Fig. 2.** Average precision-recall curves

4 Conclusion

In this paper, we have introduced a freely available web application P3S designed for similarity search in protein structure databases. The application obtains a protein structure as its input and returns the set of most similar structures. The result set can be browsed interactively or it can be downloaded as a zip archive. The application employs the SProt measure and an access method, both developed in our research group. The measure and the access method achieve very good results in comparison with other methods as was shown.

Acknowledgement. This work was supported by the Grant Agency of Charles University [project Nr. 430711]; the Czech Science Foundation [project Nr. 202/11/0968]; the Federation of European Biochemical Societies [Short-Term Fellowship]; and the Specific Academic Research [project Nr. SVV-2012-265312].

References

1. Orengo, C.A., Michie, A.D., Jones, S., Jones, D.T., Swindells, M.B., Thornton, J.M.: CATH—a hierachic classification of protein domain structures. *Structure* (London, England: 1993) 5(8), 1093–1108 (1997)
2. Meslamani, J., Rognan, D., Kellenberger, E.: sc-PDB: a database for identifying variations and multiplicity of 'druggable' binding sites in proteins. *Bioinformatics* 27(9), 1324–1326 (2011)
3. Berman, H.M., Westbrook, J.D., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E.: The Protein Data Bank. *Nucleic Acids Res.* 28(1), 235–242 (2000)

4. Zhu, J., Weng, Z.: FAST: a novel protein structure alignment algorithm. *Proteins* 58(3), 618–627 (2005)
5. Sacan, A., Toroslu, H.I., Ferhatosmanoglu, H.: Integrated search and alignment of protein structures. *Bioinformatics* 24(24), 2872–2879 (2008)
6. Shindyalov, I.N., Bourne, P.E.: Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Eng.* 11(9), 739–747 (1998)
7. Lo, W.C., Huang, P.J., Chang, C.H., Lyu, P.C.: Protein structural similarity search by Ramachandran codes. *BMC Bioinformatics* 8 (2007)
8. Tung, C.H.H., Huang, J.W.W., Yang, J.M.M.: Kappa-alpha plot derived structural alphabet and BLOSUM-like substitution matrix for rapid search of protein structure database. *Genome Biol.* 8(3), R31 (2007)
9. Aung, Z., Tan, K.L.: Rapid 3D protein structure database searching using information retrieval techniques. *Bioinformatics* 20(7), 1045–1052 (2004)
10. Chothia, C., Lesk, A.M.: The relation between the divergence of sequence and structure in proteins. *The EMBO Journal* 5(4), 823–826 (1986)
11. Galgonek, J., Hoksza, D., Skopal, T.: SProt: sphere-based protein structure similarity algorithm. *BMC Proteome Science* 9(suppl. 1) S20 (2011)
12. Reinders, J.: Intel threading building blocks: outfitting C++ for multi-core processor parallelism. O'Reilly Media, Inc. (2007)
13. Chandronia, J.M.M., Hon, G., Walker, N.S., Lo Conte, L., Koehl, P., Levitt, M., Brenner, S.E.: The ASTRAL Compendium in 2004. *Nucleic Acids Res.* 32(Database issue), D189–D192 (2004)
14. Murzin, A.G., Brenner, S.E., Hubbard, T., Chothia, C.: SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.* 247(4), 536–540 (1995)
15. Konc, J., Janezic, D.: ProBiS: a web server for detection of structurally similar protein binding sites. *Nucleic Acids Res.* 38(Web-Server-Issue), 436–440 (2010)
16. Zhang, Z.H., Bharatham, K., Sherman, W.A., Mihalek, I.: deconSTRUCT: general purpose protein database search on the substructure level. *Nucleic Acids Res.* 38(Web-Server-Issue), 590–594 (2010)
17. Krissinel, E., Henrick, K.: Secondary-structure matching (SSM), a new tool for fast protein structure alignment in three dimensions. *Acta Crystallographica Section D* 60(12 pt. 1), 2256–2268 (2004)
18. Gibrat, J.F., Madej, T., Bryant, S.H.: Surprising similarities in structure comparison. *Current Opinion in Structural Biology* 6(3), 377–385 (1996)
19. Lo, W.C., Lee, C.Y., Lee, C.C., Lyu, P.C.: iSARST: an integrated SARST web server for rapid protein structural similarity searches. *Nucleic Acids Research* 37(Web-Server-Issue), 545–551 (2009)
20. Yang, J.M.M., Tung, C.H.H.: Protein structure database search and evolutionary classification. *Nucleic Acids Research* 34(13), 3646–3659 (2006)
21. Holm, L., Rosenström, P.: Dali server: conservation mapping in 3D. *Nucleic Acids Research* 38(Web-Server-Issue), 545–549 (2010)
22. Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* 25(17), 3389–3402 (1997)

On the Parallelization of the SProt Measure and the TM-Score Algorithm

Jakub Galgonek, Martin Kruliš, and David Hoksza

Departement of Software Engineering, Charles University in Prague
Malostranske nám. 25, 118 00 Praha 1, Czech Republic
{galgonek,krulis,hoksza}@ksi.mff.cuni.cz

Abstract. Similarity measures for the protein structures are quite complex and require significant computational time. We propose a parallel approach to this problem to fully exploit the computational power of current CPU architectures. This paper summarizes experience and insights acquired from the parallel implementation of the SProt similarity method, its database access method, and also the wellknown TM-score algorithm. The implementation scales almost linearly with the number of CPUs and achieves 21.4× speedup on a 24-core system. The implementation is currently employed in the web application <http://siret.cz/p3s>.

Keywords: protein structure similarity, parallel, optimization.

1 Introduction

Measuring the similarity of protein structures has many applications in various fields of computational proteomics. Based on the hypothesis that the proteins sharing similar structure often share also other properties, the measurement of structural similarity could be used, for example, to study biological functions of proteins [1]. It could be also used to identify related proteins from the evolutionary point of view [2]. Structural similarity can reveal distant protein relations even in cases which cannot be recognized on the basis of sequential similarity since evolution tends to preserve the structure (and thus the function) of the protein rather than its sequence [3].

Computing the measure of structural similarity could be very time consuming. This computational cost become very important especially in search queries to the protein database when similarity between a query protein structure and many structures from the database must be computed.

Some of the approaches are trying to avoid this problem by designing measure along with an efficient indexing method. The *ProtDex2* method [4] represents protein structure as a set of feature vectors describing spatial properties of secondary structure element pairs. Feature vectors of the query are compared with feature vectors of the database proteins which are organized in inverted file. Hence, the time complexity of the search algorithm is dependent on the number of different feature vectors rather than on the number of structures in the database. Another approach presents the *3D-Blast* method [5]. It defines a

small structural alphabet and corresponding substitution matrix, which allows to employ the well-established algorithms of the BLAST sequence method. The *Vorometric* method [6] tries to utilize metric access methods. It represents each amino acid by a so-called *contact string*, for which the metric similarity is defined. Contact strings from the query are searched in the database of contact strings and found hits are used as seeds that are extended into alignments.

Combining the design of similarity measure with the design of indexing and access method allows the creation of efficient search system. However, this approach may also lead to some restrictions that cause a decrease in effectiveness of the similarity measure. Therefore, we have designed our similarity measure *SProt* [7] without any type of indexing method and the indexing method based on metric properties was added after the measure was designed. Even though the indexing improves performance significantly, the methods is still quite slow for real-time applications. We have employed parallelism in order to speed up the search process further.

As the processors developed in time, we have been allowed to compute larger and more complex biological tasks. The processor frequency has grown steadily for a few decades. This trend has reached an impasse recently due to the physical limitations of the silicon-based chips. The CPU development turned and focused on parallel processing instead. This major change in hardware architectures forced us to adjust our approach to high-performance computing as we can no longer rely solely on the compiler to produce optimal code for the processor. In order to achieve optimal performance, programs and algorithms must be rewritten in a way that embraces the parallel nature of the current processor architectures.

The paper is organized as follows. We briefly revise the principles and algorithms of SProt and TM-score measures (Section 2). Our parallel implementation is described in Section 3. Section 4 discusses the parallelization benefits from the perspective of experimental results and Section 5 concludes the paper.

2 Methods and Algorithms

Our method consist of two parts – the similarity measure and the access method that provides means for performing the similarity search on a database of protein structures. We will briefly describe both parts in the following section.

2.1 SProt Measure

Similarity measure SProt [7] is based on comparing local spacial neighborhoods of the amino acids present in a protein. A neighborhood of an amino acid is defined by amino acids present in a sphere with center in the given amino acid (called central amino acid) and with radius 9Å. Defining similarity on these small parts of the protein structure is much easier than working with the structure as whole. Continuous parts of the amino acid backbones included in the neighborhoods that span over central amino acids of the spheres are short enough so they

can be aligned without gaps. This will provide sufficiently long seed alignment from which the local superposition can be computed. The superposition is then used to align remaining amino acids present in the neighborhoods. This local alignment expresses the similarity of the neighborhoods.

Global alignment for whole protein structures is defined consequently by application of Needleman-Wunsch dynamic programming algorithm [8]. It uses similarity of amino acid neighborhoods as scoring function in combination with constant gap penalty model. Finally, the alignment quality is measured by well-established TM-score algorithm [9]. The algorithm tries to find the superposition that maximizes the following formula:

$$\text{TM-score} = \frac{1}{L_T} \sum_{i=1}^{L_A} \frac{1}{1 + \left(\frac{d_i}{d_0(L_T)}\right)^2}, \quad (1)$$

where L_A is the length of the alignment, L_T is the size of the query structure, d_i is the distance of the i^{th} pair of the aligned amino acids according to given superposition, and $d_0(L_T)$ is a scale function.

The TM-score expresses the quality of the alignment very well and the TM-score algorithm is quite resistant to misaligned pairs (unlike, for example, the RMSD method for which the outliers presents serious problem). However, in contrast to RMSD [10], there is no efficient algorithm that would allow us to find the exact superposition maximizing the formula quickly. Therefore, the suboptimal heuristic approach is the only solution.

The heuristic algorithm is based on the idea that two at least partially similar protein structures are likely to have structure subsets that will be very similar. For such subsets the RMSD superposition would not be much different from optimal TM-superposition. Therefore, the algorithm tries computing RMSD superpositions for various subsets of the alignment called *cuts*. A TM-score value is computed for each RMSD superposition and the maximal value is returned as the result of the algorithm.

Algorithm 1. TM-score algorithm

```

Input:  $X \in (\mathbb{R}^3)^L, Y \in (\mathbb{R}^3)^L, L \in \mathbb{N}$ 
Result:  $score \in \mathbb{R}, U_{max} \in \mathbb{R}^{3 \times 3}, T_{max} \in \mathbb{R}^3$ 
Parameters:  $max_c = 20$ 

1 begin
2    $score_{max} \leftarrow 0$ 
3   foreach  $l \in \{L, L/2, L/4, \dots, 4\}$  and  $i \in \{0, \dots, L-l\}$  do
4      $cut \leftarrow \{i, \dots, i+l-1\}$ 
5     for  $c = 0$  to  $max_c$  do
6        $(U, T) \leftarrow \text{RMSD}(X[cut], Y[cut])$ 
7        $score \leftarrow \text{TM}(X, UY + T)$ 
8       if  $score > score_{max}$  then  $score_{max} \leftarrow score; (U_{max}, T_{max}) \leftarrow (U, T)$ 
9        $cut' \leftarrow cut; cut \leftarrow \{\}$ 
10      for  $i = 0$  to  $L-1$  do
11        if  $|X[i] - (UY[i] + T)| < 3d_0$  then  $cut \leftarrow cut \cup \{i\}$ 
12      if  $cut = cut'$  then break
13
Result:  $score_{max}, (U_{max}, T_{max})$ 

```

Algorithm 2. LAESA access method

```

Input:  $q \in \mathcal{D}, k \in \mathbb{N}$ 
Result:  $R \subset \mathcal{D}$ 
Data:  $D \subset \mathcal{D}, P \subset D, d_i : D \times P \rightarrow \mathbb{R}_0^+$ 
1 begin
2    $S \leftarrow D; R \leftarrow \{\}; \forall o \in D : e(o) \leftarrow 0; threshold \leftarrow \infty; s \leftarrow$  arbitrary member of  $P$ 
3   while  $S \neq \{\}$  do
4      $S \leftarrow S \setminus \{s\}$ 
5      $dist(s) \leftarrow d(q, s)$                                 // distance measuring
6      $R \leftarrow k\text{-nearest objects from set } R \cup \{s\}$ 
7     if  $|R| = k$  then  $threshold \leftarrow \max\{dist(o); o \in R\}$ 
8      $d_s \leftarrow \infty$ 
9     foreach  $o \in S$  do                                // approximation and estimation loop
10    if  $s \in P$  then  $e(o) \leftarrow \max\{|d_i(s, o) - dist(s)|, e(o)\}$       // approximation
11    if  $e(o) > threshold$  then  $S \leftarrow S \setminus \{o\}$                       // elimination
12    if  $e(o) < d_s$  then  $d_s \leftarrow e(o); s' \leftarrow o$ 
13
14    $s \leftarrow s'$ 
15
16 Result:  $R$ 

```

The TM-score method is outlined in Algorithm 1. It receives two sequences of coordinates X and Y , of the length L , representing coordinates of aligned pairs of amino acids for the first and the second protein respectively. The result is the value of computed TM-score and the corresponding superposition. The *outer loop* (lines 3-12) of the algorithm takes all continuous parts of the alignment of lengths $L, L/2, \dots, 4$ as the initial cuts. Each iteration of the *internal loop* (lines 5-12) computes the superposition for the current cut (line 6) and constructs a new cut that contains only those amino acid pairs that are closer than the $3d_0$ threshold after the superposition (lines 10-11). The internal loop continues until the cut is stabilized or the maximal number of iterations is reached. After that the outer loop continues by picking another initial cut for testing.

2.2 SProt Access Method

The similarity measure SProt achieved excellent results at information retrieval experiments. However, the procedure is very time consuming. We have designed specific access method based on LAESA metric indexing [11,12] in order to increase the performance of the measure.

Metric access methods [13] are being used for similarity search that employs similarity measure d (called distance), that complies with metric axioms. These axioms allow us to estimate the distance lower bounds between two objects based on known distances to another objects. Let us have query object q and database object o for which we are trying to determine their similarity. If we precompute the distance between o and another database object p (usually called pivot) and the distance from q to p is computed as well, we can use triangle inequality to estimate the lower bound for the distance between q and o :

$$d(q, o) \geq |d(q, p) - d(o, p)| \quad (2)$$

If the estimated lower bound is greater than some predefined range or the distances to the best result objects found so far, we can quickly dismiss object o

Table 1. Required computational time

Method Subpart	Time (min:sec)	Relative Time (in percents)
Query Model Loads	0:03	0.06%
Alignments	96:12	85.91%
Alignments: Score Matrix	95:43	85.48%
TM-score	13:41	12.22%
LAESA: Approximation and Estimation	2:01	1.81%
Total Query Evaluation	111:58	100.00%

without computing potentially expensive distance $d(q, o)$ as the o will certainly not be present in the results.

The same approach employs the LAESA metric access method. The method is being used to find k database structures that are the most similar to the query structure. It takes a subset (called *pivots*) of structures $P \subseteq D$ from given database D and precomputes table d_i of distances $d(p, o)$ for each pivot $p \in P$ and database structure $o \in D$. When the distance between query structure q and any of the pivots p is computed during the query evaluation process, this distance can be used to update lower bound estimates for the remaining structures and prune them using triangular inequality described earlier.

The fundamentals of this method are summarized in the Algorithm 2. It expects only the query structure q and number k (beside the database itself) as input and yields closest k structures from the database as output. The algorithm iteratively processes a set of candidate structures S and keeps a set R of intermediate results (closest k structures). Initially the S contains the whole database D and the algorithm stops when S becomes empty. The algorithm also maintains an array of lower bound estimates $e(o)$, that contains the greatest lower bound estimate for each candidate o . In each iteration (lines 3-12), a structure $s \in S$ with the lowest $e(s)$ value is taken and also removed from S (line 4). The distance $d(q, s)$ is computed (line 5) and intermediate results in R are updated accordingly (line 6). If s is a pivot ($s \in P$), the lower bound estimates $e(o)$ for the candidates remaining in S are updated using values from the precomputed pivot table and the triangle inequality (line 10). If any estimate $e(o)$ is greater than threshold value $t = \max\{d(q, x), x \in R\}$, structure o is pruned from S (line 11). When the algorithm terminates, set R contains the final result.

Since the SProt measure does not conform to all metric axioms, the access method has to be slightly modified and works only as an approximation [7]. The lower bound estimation formula has been altered for asymmetric measures and some new attributes have been incorporated into the method to increase precision of measures that does not fully conform to the triangle inequality axiom. However, the main idea of the algorithm was preserved.

3 Parallel Implementation

There are many approaches to the parallel processing of our task. The most straightforward would be to process multiple queries concurrently. In such case,

each query is evaluated by separate task while tasks are processed concurrently by available processors, however, the task itself is processed in serial manner. This approach is very easy to implement and, since the tasks are completely independent, it is also very efficient. Unfortunately, sufficient number of queries must be submitted to the system at the same time in order to occupy all available processors. Furthermore, if there are a few long lasting queries amongst a block of short queries, the system will be left with a few big tasks that will linger (occupying only one core each) while remaining processors become idle.

Second possible approach is to modify the method itself, so that one query evaluation utilizes multiple cores. We have profiled our code to determine, how long does each part of the algorithm take. Times measured in the profiling run and their relative contribution are summarized in Table 1. The ProtDex2 dataset [4] containing 34,055 structures and 108 queries was used for the profiling. The search yielded 20 the most similar structures for each query.

The profiling results clearly show that the alignment computation (especially the score matrix computation) time dominates the entire evaluation. Therefore, this part would benefit the most from the concurrent execution. However, according to Amdahl's law [14], serial parts create serious scalability issues, thus we parallelize the TM-score computation and the LAESA approximation and estimation as well. We do not parallelize the query model loading as the I/O operations are serial in nature and the loading itself takes insignificant part of the overall time.

We can also choose completely different method and try a data parallel approach. In such case, the database is divided into fragments, which are treated as independent (small) datasets and queried concurrently. Each fragment yields a local result and local results of all fragments are merged into the final result. Even though this method has minimal scheduling overhead and does not depend on the number of queries available, it is not suitable for our algorithm. Our search measure employs the LAESA method to prune the number of computed distances (structure similarities). This method performs better on larger datasets and with larger number of pivots. If the database fragments are too small, the effect of the LAESA is significantly diminished. In the worst case, it can deteriorate into simple scan of the entire database. For these reasons, we have abandoned this approach and focus solely on the previous two approaches.

Concurrent Alignment Computation. According to profiling results, the alignment algorithm spends its almost entire time by computing the scoring matrix. Since each item of the matrix is computed independently, the computation can be easily modified to use two-dimensional parallel for-loop instead of its sequential version. The matrix is evenly divided into tiles and each tile is processed as a concurrent task. The size of the tiles is chosen carefully so that one task computing one tile will consist of at least 10^5 CPU instructions¹. The tile-wise approach is preferred to both row-wise and column-wise approaches as it is better optimized for CPU caches.

¹ Tasks containing 10^5 to 10^6 of CPU instructions were observed as optimal [15].

Concurrent TM-Score Computation. In the case of TM-score, we chose to parallelize the outer-loop only. Internal loops are rather short, thus, the overhead of task scheduling would be too great. The outer loop iterates over initial cuts. An initial cut can be defined by its length and the index of the first pair of aligned amino acids (we denote the length-index tuple the *initial configuration*). Our approach first generates all initial configurations to an array and then traverses the array by the parallel for-loop. Hence, each configuration is processed independently as a separate task. In order to eliminate explicit synchronization, each thread keeps its own maximum value of computed scores and the corresponding superposition. The best score is picked from the local values when the parallel for-loop terminates. Concurrent computation of initial cuts introduces nondeterminism into the process, since we arbitrarily change the order of score comparisons. Even though this does not affect the method, a deterministic approach is preferred by the users and required for the verification of our parallel implementation. We have modified the score comparison, so that the configuration index is used as secondary key in case two score values are equal. This modification ensures that the choice of the best superposition is deterministic despite the concurrent processing.

Our implementation generates the initial configurations sequentially and then processes them concurrently. In theory, it would be better to generate the configurations by one thread and dispatch them immediately to other available threads. We can simply utilize a parallel while-loop [15] provided with an initial configuration generator. However, this approach requires significantly more thread synchronization, thus having greater overhead. Empirical results show that it is not as efficient as concurrent processing of configurations, which have been pregenerated sequentially.

LAESA Approximation and Estimation Loop. The main loop of the LAESA method cannot be effectively parallelized since each iteration updates the set of candidates as well as lower bound estimates which are required by next iteration. Fortunately, we are still able to parallelize the internal loop, called also the *approximation and estimation loop* (see Algorithm 2: lines 9-11). The internal loop updates the estimate values $e(o)$, filters the candidate set S , and finds the candidate s with the lowest estimate $e(s)$. To avoid synchronization, the threads work on different parts of S (and $e(o)$ respectively). When the candidate s needs to be found, each thread t has its own candidate s_t and after the entire set S is processed concurrently, the candidate s is quickly found from the set $s \in \{s_t | t \in \text{Threads}\}$. Since we are parallelizing the internal loop, a barrier must be present at the end of the main loop. Therefore, the whole process is slightly less efficient in comparison with other parts of the algorithm.

4 Discussion

The experiments were performed on three different versions of the parallel implementation (all based on the *Intel Threading Building Blocks* library [15]) and

compared with the performance of the serial implementation. The first one (denoted *Query-Parallel*) employs the first parallelization approach – queries are evaluated concurrently, but each query runs serially. The second one (*Intern-Parallel*) utilizes the second approach to parallelism – each query is parallelized inside, but queries are submitted for evaluation sequentially. The third one (*Full-Parallel*) combines previous two methods to achieve even higher scalability.

Each of these versions points out different aspects of the parallel processing. The Query-Parallel version is the most efficient since it reduces the overhead of task spawning, scheduling, and synchronization. However, it presumes that there are enough tasks to keep available CPU cores occupied. The Intern-Parallel implementation better reflects the real world situation when only one query is being processed and we still want to exploit parallelism to expedite the evaluation of this query. Combination of these two methods provides the ultimate solution (Full-Parallel) that reaches the limits of the optimal scalability.

We measure the efficiency of parallelization by the speedup factor:

$$S_p = \frac{T_{seq}}{T_p}, \quad (3)$$

where T_{seq} is the execution time of the sequential algorithm and T_p is the execution time of the parallel version running on p CPU cores. Note that we do not distinguish between two physical CPUs and two CPU cores on one die. Even though that there are some differences like NUMA factor or L3 cache-sharing issues, these hardware properties had negligible impact on overall performance in our case. On the other hand, we do not consider two logical cores mapped to one physical core using hyper-threading technology to be independent cores and all tests were performed on independent physical cores only.

We have used the Dell M910 server containing four Intel Xeon E7540 CPUs, six cores clocked at 2.0 GHz each (i.e., 24 cores total). The server was equipped with 128 GB of RAM organized as 4-node cache coherent NUMA. The experiments were conducted using the ProtDex2 dataset [4] containing 34,055 structures and 108 testing queries. The speedup of all three implementations measured for up to 24 cores is depicted in Figure 1. We have used an approximation curve that demonstrates the speedup of the algorithm. The approximation curve minimizes the squared differences from the measured values.

Figure 1 (left part) indicates that Query-Parallel version exhibits lower scalability and larger variance of measured results than other versions. This is due to large granularity of the tasks (as we execute only 108 queries), thus the performance strongly depends on the task distribution over available threads. It is safe to assume that the performance will be better if the number of the tasks is significantly larger than number of available processors. The Intern-Parallel version performs better than Query-Parallel as it produces more fine-grained tasks. However, this approach still suffers from the synchronization between two subsequent queries – evaluation of one query must be completed before another query is started. These synchronization points between queries create performance gaps when CPUs are heavily underutilized, thus the speedup is still less than optimal ($16\times$ on 24 cores). On the other hand, the Intern-Parallel version

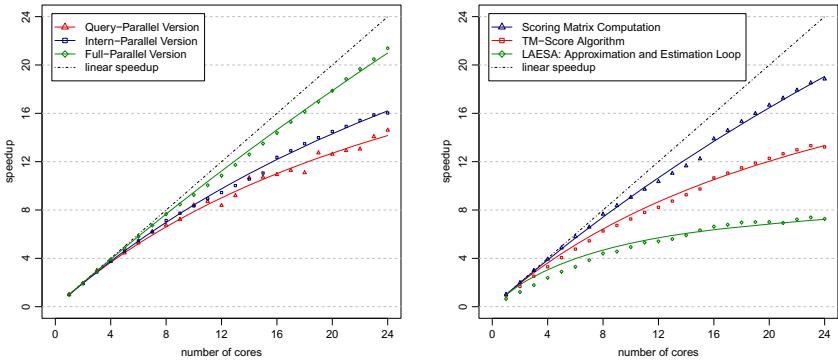


Fig. 1. Measured speedup of the implementation – comparison of the versions on the left and the algorithm parts overview on the right

simulates the situation when the queries are provided by the user in sequential manner. We have observed that significant speedup can be achieved on multiple cores even when single query is processed.

If we combine both approaches, we preserve the benefit of fine-grained tasks and we eliminate the synchronization points between queries. The Full-Parallel version reached $21.4\times$ speedup on 24 cores. Considering the inevitable overhead of the task scheduling and the presence of data loading parts which are sequential in nature, it is safe to say that the scalability of the Full-Parallel version is almost optimal. This version is particularly useful when multiple users search the database and we use it in our web application, which is available at <http://siret.cz/p3s>.

To compare the efficiency of each part of the parallel implementation, we have also measured their individual speedups (see Figure 1, right part). The results match our expectations. The scoring matrix computation is the most independent, thus exhibits the best speedup. The TM-score algorithm, which is often adopted by other similarity methods, scales also exceptionally well. The weakest part of the algorithm is the LAESA method since it is iterative and only the internal loop was parallelized. We have observed smaller speedup due to synchronization within the main loop and small size of the tasks being dispatched to the threads.

5 Conclusion

We have compared three approaches to the parallel implementation of the SProt measure and its access method. Our best version reached excellent speedup and scales almost linearly with the number of CPU cores. Furthermore, we believe, that achieved speedup is almost optimal, since there is some inevitable overhead and sequential parts that cannot be parallelized. We have also presented concurrent version of the TM-score superposition algorithm. Since the TM-score

is being used in other methods as well, its concurrent version may find other applications beyond the realms of the protein structure similarity measures.

Acknowledgement. This work was supported by the Grant Agency of Charles University [project Nr. 430711]; the Czech Science Foundation [project Nr. 202/11/0968]; and the Specific Academic Research [project Nr. SVV-2012-265312].

References

1. Orengo, C.A., Michie, A.D., Jones, S., Jones, D.T., Swindells, M.B., Thornton, J.M.: CATH—a hierachic classification of protein domain structures. *Structure* (London, England: 1993) 5(8), 1093–1108 (1997)
2. Balaji, S., Srinivasan, N.: Use of a database of structural alignments and phylogenetic trees in investigating the relationship between sequence and structural variability among homologous proteins. *Protein Eng.* 14(4), 219–226 (2001)
3. Chothia, C., Lesk, A.M.: The relation between the divergence of sequence and structure in proteins. *The EMBO Journal* 5(4), 823–826 (1986)
4. Aung, Z., Tan, K.L.: Rapid 3D protein structure database searching using information retrieval techniques. *Bioinformatics* 20(7), 1045–1052 (2004)
5. Tung, C.H.H., Huang, J.W.W., Yang, J.M.M.: Kappa-alpha plot derived structural alphabet and BLOSUM-like substitution matrix for rapid search of protein structure database. *Genome Biol.* 8(3), R31 (2007)
6. Sacan, A., Toroslu, H.I., Ferhatosmanoglu, H.: Integrated search and alignment of protein structures. *Bioinformatics* 24(24), 2872–2879 (2008)
7. Galgonek, J., Hokszá, D., Skopal, T.: SProt: sphere-based protein structure similarity algorithm. *BMC Proteome Science* 9(suppl. 1), S20 (2011)
8. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48(3), 443–453 (1970)
9. Zhang, Y., Skolnick, J.: Scoring function for automated assessment of protein structure template quality. *Proteins* 57(4), 702–710 (2004)
10. Kabsch, W.: A solution for the best rotation to relate two sets of vectors. *Acta Crystallogr. A* 32(5), 922–923 (1976)
11. Micó, M.L., Oncina, J., Vidal, E.: A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters* 15(1), 9–17 (1994)
12. Moreno-Seco, F., Micó, L., Oncina, J.: Extending LAESA Fast Nearest Neighbour Algorithm to Find the k Nearest Neighbours. In: Caelli, T.M., Amin, A., Duin, R.P.W., Kamel, M.S., de Ridder, D. (eds.) *SSPR and SPR 2002*. LNCS, vol. 2396, pp. 718–724. Springer, Heidelberg (2002)
13. Chávez, E., Navarro, G., Baeza-Yates, R.A., Marroquín, J.L.: Searching in metric spaces. *ACM Comput. Surv.* 33(3), 273–321 (2001)
14. Amdahl, G.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18–20, 1967, Spring Joint Computer Conference, pp. 483–485. ACM (1967)
15. Reinders, J.: Intel threading building blocks: outfitting C++ for multi-core processor parallelism. O'Reilly Media, Inc. (2007)

Stochastic Simulation of the Coagulation Cascade: A Petri Net Based Approach

Davide Castaldi¹, Daniele Maccagnola¹, Daniela Mari^{2,3}, and Francesco Archetti^{1,4}

¹ DISCo, University of Milan-Bicocca, Viale Sarca 336, Milan, 20126, Italy

² IRCCS Cà Granda Ospedale Maggiore Policlinico Foundation,
Via Pace 9, Milan, 20122, Italy

³ Department of Clinical Sciences and Community Health, University of Milan, 20122, Italy

⁴ Consorzio Milano Ricerche, Via Cozzi 53, Milan, 20126, Italy

{davidefabio.castaldi,daniele.maccagnola}@disco.unimib.it,
daniela.mari@unimi.it, francesco.archetti@unimib.it

Abstract. In this paper we developed a Stochastic Petri Net (SPN) based model to introduce uncertainty to capture the variability of biological systems. The coagulation cascade, one of the most complex biochemical networks, has been widely analyzed in literature mostly with ordinary differential equations, outlining the general behavior but without pointing out the intrinsic variability of the system. Moreover, the computer simulation allows the assessment of the reactions over a broad range of conditions and provides a useful tool for the development and management of several observational studies, potentially customizable for each patient. We describe the SPN model for the Tissue Factor induced coagulation cascade, more intuitive and suitable than models hitherto appeared in the literature in terms of bioclinical manageability. The SPN has been simulated using Tau-Leaping Stochastic Simulation Algorithm, and in order to simulate a large number of models, to test different scenarios, we perform them using High Performance Computing. We analyze different settings for model representing the cases of “healthy” and “unhealthy” subjects, comparing their average behavior, their inter- and intra-variability in order to gain valuable biological insights.

Keywords: Systems Biology, Coagulation, Stochastic Simulation, Petri Nets.

1 Introduction

Petri Nets are a modeling tool widely used to represent biological systems, thanks to their intuitive graphical representation which helps to model and understand even systems with a complex structure. The graphical aspects of the Petri net are quite similar to biochemical network representation, and this gives superior communication ability to models and facilitates their design.

Moreover, modelling biological systems requires to take into account uncertainty in system behaviors, due to external interferences (drugs, environment, etc), inter-variability (personal features) and intra-variability (intrinsic noise and low number of

molecules) of complex systems. Using the Petri Nets extension, Stochastic Petri Nets (SPNs), we can describe these random aspects (Goss and Peccoud, 1998; Srivastava, Peterson and Bentley, 2001).

A biological pathway which has recently attracted strong interest for its potential to augment bioclinical knowledge in several therapeutic domains, is the coagulation cascade (Chatterjee et al., 2010; Wajima, Isbister and Duffull, 2009). This model has been deeply investigated, resulting in complex networks where two sub-pathways, *Intrinsic* and *Extrinsic* interact in a *Common* pathway to produce active fibrin which affects the haemostasis process (Butenas and Mann, 2002). Both pathways are required for normal hemostasis and there are different feedback loops between the two pathways that amplify reactions to produce enough fibrin to be able to crosslink each other to form a fibrin clot.

Modulation of extrinsic (Tissue Factor (TF) Pathway) and intrinsic (Contact Pathway) pathways is affected either by several positive and negative feedback mechanisms, or by a set of natural proteins that restrict coagulation progression. In healthy physiological conditions constant generation of small amounts of coagulation active factors (active proteases) effects the constitutive haemostasis, while another biological mechanism, the fibrinolytic system, counterweights the process. All together, coagulation, anticoagulation and fibrinolysis define a delicate physiological balance, named “Haemostatic Balance” (Gaffney, Edgell and Whitton, 1999); significant deviations from this equilibrium, hypercoagulability or hypocoagulability unbalance, can evolve in cardiovascular adverse events.

Most of the existing papers model coagulation as a system of ordinary differential equations (ODEs), considering single complex, partial pathways, cell based model, comprehensive humoral coagulation network, or with a PDEs approach to model rheological behavior of molecules, blood flow. As pointed out by Danforth et al. 2009, these approaches cannot express the variability of the coagulation pathway, which is widely validated both in wet lab experiments and in vivo system (Couris et al. 2006). This aspect has been considered by a few works using different techniques, ranging from sensitivity parameter analysis (Luan et al. 2007), ODEs with stochastic coefficients (Corlan and Ross 2011), stochastic differential equation of dissipative particle dynamics method (Filipovic, Kojic and Tsuda 2008) and probability distributions evaluation of reactions for sophisticated stochastic simulations (Lo et al., 2005). However, none of these stochastic approaches allows a detailed analysis of the coagulation system whose results can be matched with existing clinical tests, in particular the “Prothrombin Time” (PT) test.

In this paper the authors developed a model of the coagulation, represented by counterbalance of coagulation and anticoagulation proteins, based on a Petri Net modeling framework (Sec.2). This model offers an intuitive graphical representation of TF pathway with a manageable modularity. The intrinsic and extrinsic pathways have been evaluated as separate entities to match *in vitro* test results, but the *in vivo* analysis requires to consider both of them during the simulation. The presented model represents a new stochastic bioclinical approach of modelization of the extrinsic pathway, which implies the action of the intrinsic way in order to have model applicable to clinical analysis.

This work is focused on comparing the behaviour of a “healthy” subject with that of an “unhealthy” subject, starting from a Stochastic Petri Net (SPN) model and changing the initial marking to represent the effect of a prothrombotic event. The models will be analyzed using simulation: first we will use a deterministic approach to tune the model parameters, converting the SPN into a continuous Petri Net (CPN) and solving the related system of ODEs (Sec.3); second, we will perform the actual analysis with stochastic simulation, employing Gillespie’s Stochastic Simulation Algorithm (Sec.4).

It is clear that the stochastic model is more effective than its deterministic counterpart in unearthing valuable biological insights, but its computational demands can become prohibitive even for moderate size networks. Approximate methods, along with the use of high performance computing (HPC) are shown in Sec.5 to bring the computational complexity of the stochastic approach within manageable limits.

2 Petri Nets and Biochemical Pathways

A Petri Net is a weighted, directed and bipartite graph, originally conceived for modeling systems with interacting concurrent components and now widely used to study complex model such as biochemical pathways. This paper employs notation used in (Heiner, Gilbert and Donaldson, 2008). In terms of a chemical reaction network, coagulation involves a sequence of highly connected concurrent processes with many simultaneous positive and negative feedback loops that specify the beginning, the progression and the amplification of whole system.

The graphical representation of the Petri net is quite similar to a biochemical network: places represent compounds (such as metabolites, enzymes, cofactors etc.) participating in a reaction, which is represented by a transition. The arcs which connect places and transitions will represent the stoichiometry of the reaction, while the tokens associated to a place will represent the amount of molecules of that particular reactant.

The most abstract representation of a biochemical network is qualitative and is minimally described by its topology by using a place-transition Petri Net (p-t PN, or qualitative Petri Nets, QPN).

Using this formalism, we have been able to build up a model of the coagulation pathway. In particular, the relevant role played in coagulation process by the Tissue Factor (Chu, 2011), also in cardiovascular diseases (Steffel, 2006; Mackman, 2004), has supported the idea of modeling the extrinsic pathway in details. We accurately reproduce the pathway, including all the positive and negative feedback circuits, as well as the exact number of molecules involved in the process.

The starting point in building the model is to define the initial marking and reaction rate constants, which we obtained from biological databases as Brenda (<http://www.brenda-enzymes.org>), model repositories as BioModel Database (<http://www.ebi.ac.uk/biomodels-main>), integrated with other data found in PubMed (<http://www.ncbi.nlm.nih.gov/pubmed>). The initial marking represents in our model

the average value of the observed physiological range. The reaction rates have been tuned with a trial and error approach (based on enzyme kinetic assumptions) to replicate the thrombin generation time given by a bioclinical PT test (Khanin et al., 1999), and reproduce a titration curve of the thrombin formation in a biochemical test (Buteñas, van't Veer and Mann, 1999). Another approach to approximate the reaction constants is given in (Shaw, Steggles and Wipat, 2006). In the initial marking (M_0), only 11 places have a non-zero number of tokens, which ranges from 75 to $3,01 \cdot 10^8$ molecules (the number has been deduced from the values reported in Fig.1), with a considered plasma volume of $1 \cdot 10^{-10}$ liters. The whole set of places, transitions and parameters (initial marking and rate constants) are given as supplementary material on the authors' website (www.nedd.unimib.it - Downloads - Supplementary Materials).

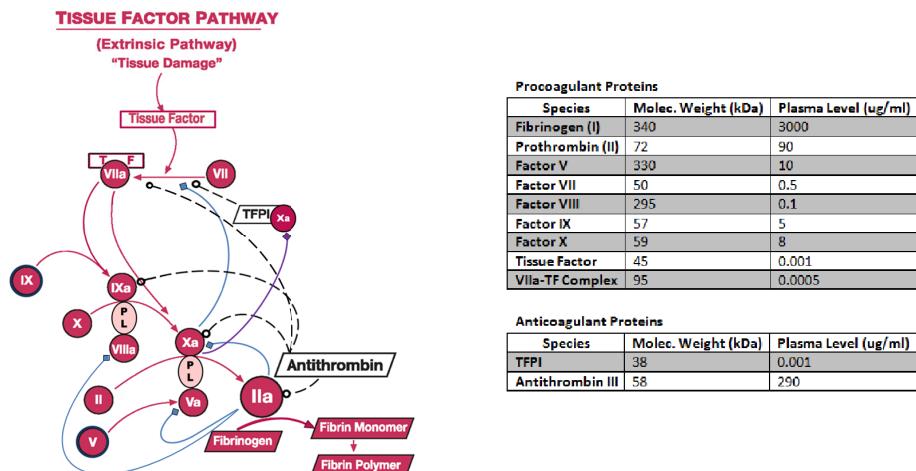


Fig. 1. A graph representation of Tissue Factor Pathway (modification of image from <http://www.enzymeresearch.co.uk>) and species physiological characteristics. The pathway can be subdivided in a procoagulant subpathway (red arcs), positive feedbacks (blue arcs), negative feedbacks (purple arcs) and inhibitor subpathways (black dashed arcs).

Fig.2 shows the Tissue Factor pathway represented as a Petri Net model and the description of all the places. We use macro-nodes to give a neat and hierarchically structured representation of our model. Fig.3 shows in detail the two types of reactions described in the hierarchical PN: complex association/dissociation (AD#) and Michaelis-Menten enzyme reaction (MM#). The hierarchical structure of the model does not change any properties of the flat model (Breitling et al., 2008). Our network is composed, in total, by 26 places and 18 macro-nodes, which can be unfolded into a network of 35 places and 43 transitions. The standard semantics for qualitative Petri nets does not associate a time with transitions or the sojourn of tokens at places, and thus these descriptions are time-free. The qualitative analysis considers however all possible behavior of the time-independent system.

Timed information can be added to the qualitative description by using an extension of PN called Stochastic Petri Nets (SPN). The SPN description preserves the discrete state description, but in addition associates a probabilistically distributed firing rate (waiting time) with each reaction. All reactions which occur in the QPN can still occur in the SPN, but the probability that a reaction will happen within a period of time depends on the associated firing rate, which is defined as a negative exponential probability density function:

$$f_{X_t} = \lambda_t(m) \cdot e^{(-\lambda_t(m) \cdot \tau)} \quad , \tau \geq 0$$

where the waiting time X_t is function of the transition rate $\lambda_t(m)$ (a *stochastic mass-action hazard function* as defined in (Heiner, Gilbert and Donaldson, 2008)) and the time τ . SPN are a specification language which enables both a deterministic solver based on ODE (through another specific called Continuous Petri Nets), and a stochastic solver based on Gillespie Stochastic Simulation Algorithm (SSA).

3 Deterministic Approach

The continuous model, represented by a Continuous Petri Net (CPN), replaces the discrete values of species with continuous values, and instantaneous firing of a transition is carried out like a continuous flow.

The semantics of a continuous Petri Net is defined by a system of ODEs, whereby one equation describes the continuous change over time of the token value in a given place, influenced by transitions which increase or decrease the token value. A Continuous PN contains all the information needed to generate the system of ODEs. Each place in CPN will determine an ODE based on its pre- and post-transitions: for example, from a simple example of Continuous PN we can generate an equation for each place by looking at the transitions directly connected to the places. The fast simulation time given by deterministic approach allows tuning the initial marking of the model in order to have a result that matches either the behavior shown in literature or in deterministic simulations. Also, this approach allows determining the average simulation time: as stochastic simulation can be time consuming, we need at least an approximation of the time span required to observe our target behaviour.

To confirm the reliability of our simulations, we compared the behavior of fibrin (Ia) and thrombin (IIa) in the mathematical model proposed by Khanin, Rakov and Kogan (1998), with the results given by the deterministic ODE solver on our model (Fig.4). The images show clearly that both products have the same trend as in Khanin's mathematical model. The deterministic approach is useful to compare the average behavior of our model to literature, but it cannot highlight other important characteristics unidentifiable in the average behavior, such as the biological systems variability. These problems will be overcome with the stochastic approach.

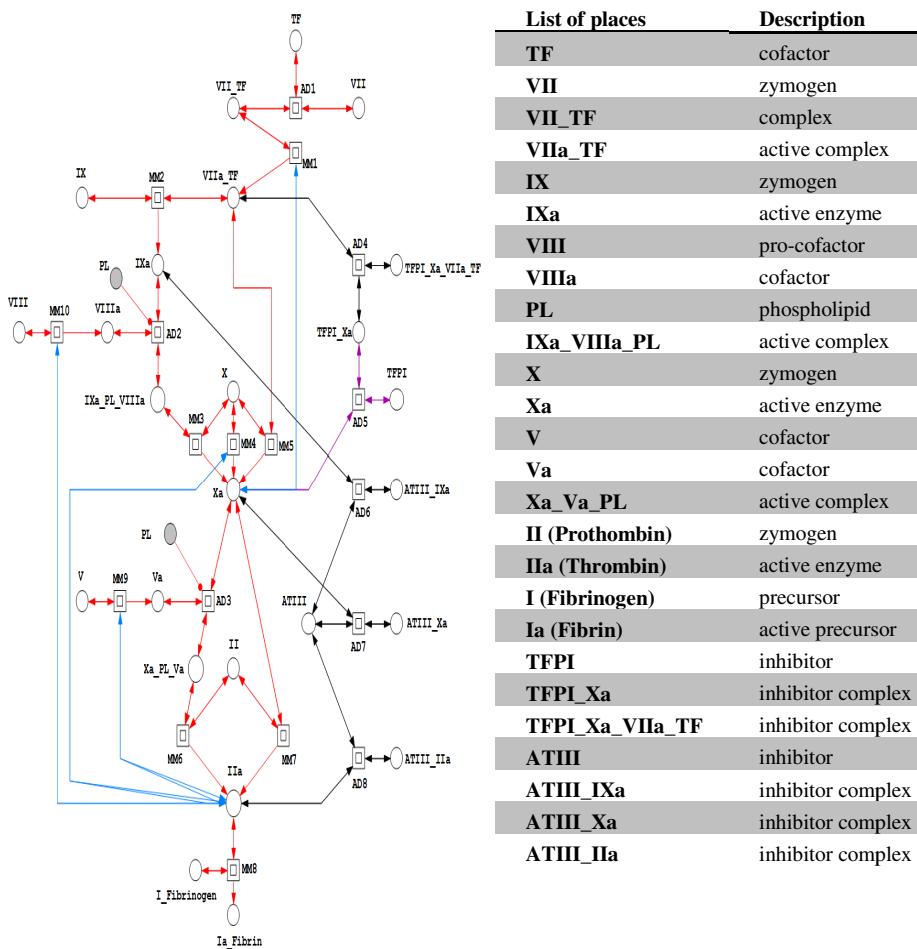


Fig. 2. High-level Petri Net model of Tissue Factor pathway, with biological description of the places

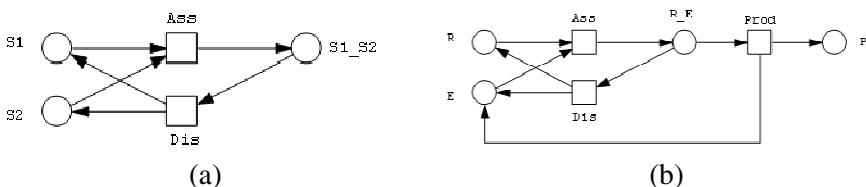


Fig. 3. Description of macro-nodes. (a) Representation of association/dissociation of a complex (AD# in high-level PN). (b) Representation of a Michaelis-Menten enzyme reaction (MM# in high-level PN).

4 Stochastic Approach

Deterministic simulation techniques assume that concentrations vary deterministically over time and that concentrations vary continuously and continually. However, these assumptions may not be valid for some important aspects of biological systems and this limitation can hamper capturing important properties of the system under study. The most important limitations of deterministic approach are the following ones:

- Although efficient in terms of computational costs, it cannot accurate for systems that contain low-rate reaction, related to species occurring in small molecular quantities. If a system contains small quantities of a species, then representing this species in molecular reactions by its average values can introduce either significant bias in model predictions or can even change the model's qualitative properties.
- When studying the behavior of bi-stable or multi stable systems, deterministic modeling (whose behaviour depends only on initial conditions) may not adequately describe the distributions of system responses, limiting to average values can cause overlooking important features of the system dynamics.
- Deterministic approach ignores stochastic variation of the system under study, which is given by the inability to represent much information about the molecules (such as position, orientation and momentum). As stochasticity in systems biology has a very important biological meaning (Cevenini et al., 2010), it is interesting to investigate the stochastic variation of system trajectories.

Stochastic simulation approach is applied in order to study effects of random fluctuations in numbers of molecular species in systems biology models.

The most common stochastic simulation algorithm (SSA) is the “Direct Method” proposed by Gillespie (Gillespie, 1977); it explicitly simulates each reaction event in the system, therefore it correctly accounts for all stochastic fluctuations. Thus, the algorithm has time complexity approximately proportional to the overall number of particles and reactions in the system. Direct Method uses two random numbers per step to compute which will be the next reaction and when it will happen (the time step τ), as described in (Gillespie 1977).

This method has the best accuracy among all Stochastic Simulation Algorithms, but it has a prohibitive time complexity for a system with a high number of reactions and particles.

In order to accelerate simulation of the coagulation model, we used a faster SSA called ‘Tau-leap method’ (Gillespie, 2001). This is faster than Direct Method, because it avoids the simulation of every single reaction event. Instead, it “leaps” along the time axis in steps of length τ , which contains many single reaction events.

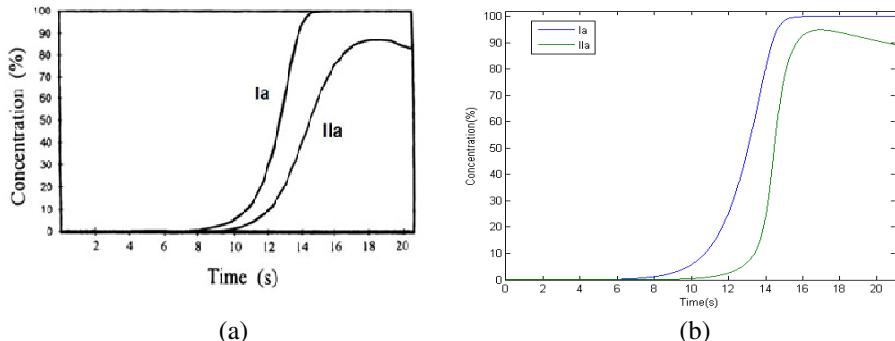


Fig. 4. Behavior of Fibrin (Ia) and Thrombin (IIa) in (Khanin, Rakov and Kogan, 1998) (a), and in our model (b)

τ must fulfill a property called “Leap Condition”, which means it has to be small enough that no significant change in the reaction rates occurs during $[t, t+\tau]$. Note that τ , differently from direct method, is not given by the reaction system, but it is fixed and user defined. Choosing the correct value for τ is a problem much discussed in literature (Gillespie, 2001; Cao and Gillespie, 2006). A single step in Tau-Leap usually takes more time than a step in Direct Method, because computing a Poisson random number is expensive. However, Tau-Leap method allowed us to simulate the coagulation model much faster than the Direct Method, because when the particle number increases, many reactions are simulated at once within one time step.

5 Experimental Results

Our main work is focused on simulating different models of haemostasis, some with healthy behavior and some with unhealthy behavior, in order to test the behavioral changes under the influence of external stimuli. For each kind of model we perform many simulations, in order to represent the real physiological variability of coagulation observable in laboratory tests. We can consider the phenomenon of variability of this system from two points of view, in which different simulation represent:

Inter-variability: different subjects we assume to have the same initial marking, that lead them to have different patterns of coagulation;

Intra-variability: a single subject, with an average initial marking, who can show different trends because of the intrinsic variability.

In both cases, performing several simulations we can observe how the variability of biological system is reproducible by stochastic simulation. This approach allows us to evaluate whether the sample paths generated by simulation follows the real behavior of the system in normal and stressed conditions, allowing the validation of our models.

5.1 Modeling and Simulation Framework

In order to perform simulation on our models, we employed two different tools:

- *Snoopy* (Rohr, Marwan and Heiner, 2010) has been used as a tool to build the models as Stochastic Petri Nets, to convert them to Continuous Petri Nets and perform deterministic simulation of the associated system of ODEs. Snoopy also allows to perform stochastic simulations using Direct Method SSA, but since in this work we based on Tau-Leaping Method, we chose to employ another tool for these simulations.
- *Dizzy* (Ramsey, Orrel and Bolouri, 2005), a biochemical kinetics simulation software package, which includes a series of tools to perform stochastic simulations of the models, in particular Tau-Leaping Method SSA.

Fig. 5 shows the scheme we followed to simulate the models. The first step is building a SPN based on the notions on coagulation, then we have two ways to simulate it: use Snoopy and generate the ODE system (based on a CPN) and solve it to get a deterministic result; export the SPN information in a SBML document, and use Dizzy simulate the model using Direct Method or Tau-Leaping Method.

At first, the simulations have been performed on a Dell Studio XPS 7100 with AMD Phenom II X6 1035T 2,60 GHz (with 6 cores) and 8 GB RAM, in order to test a small number of results and the average time requirements. After fixing the parameters of both models using deterministic approach, we have analyzed the variability of the system with a stochastic method. We tested both Direct Method SSA and Tau-Leaping Method, in order to find the best algorithm in terms of accuracy and time complexity. Direct Method takes a prohibitive time because of the size of the model. A single simulation takes 350 hours to simulate 10 seconds of coagulation, and simulating the model up to 25 seconds (the minimal time required for analyze the behaviour of the model) might take many months to complete. Tau-Leaping Method allows a faster simulation, inducing only a minor loss in accuracy. It takes around 8 hours to simulate 25 seconds of coagulation, performing one simulation for each core (for a total of six simulations in 8 hours). Our focus is on computation of an ensemble of Tau-Leaping SSA realizations, and we need at least 100 simulation results to estimate the characteristics of a model with an acceptable statistical accuracy. As our six-core system can perform only six realizations at a time, the whole process would require more than 120 hours. Thus, we decided to execute the algorithm on the Bari INFN Computer Farm (based on Torque/Maui, and composed of ~3700 CPU/Core with a disk space of ~1.3PByte), which allowed us to parallelize the process and simulate all the 100 realizations simultaneously, requiring a total of 8 to 16 hours to complete them (depending on the model and on the clock rate of the available cores). The simulations were performed on a single cluster, using up to 100 nodes to simulate all the realization simultaneously. We developed a Unix bash script which automatically modify the original (healthy model) changing its parameters (initial marking), and submit the relative jobs to the FARM. The results have been collected in .csv documents, and processed using MATLAB in order to compute the trajectories and significant values such as peak times (in particular, their mean and standard deviation).

It is important to note that we parallelize only the ensemble of SSA realizations, but each single simulation is not split on multiple processors. Most of the attempts in literature parallelized across the simulations, trying to speed up the process using GPUs (Li and Petzold, 2009) and improving the random number generation. Attempts have been made to parallelize single simulations (e.g. see Dittamo and Cangelosi (2009)), but these algorithms work by splitting the set of reactions among blocks, and the structure of our Stochastic Petri Net cannot be easily partitioned because of the strong interconnections among the nodes.

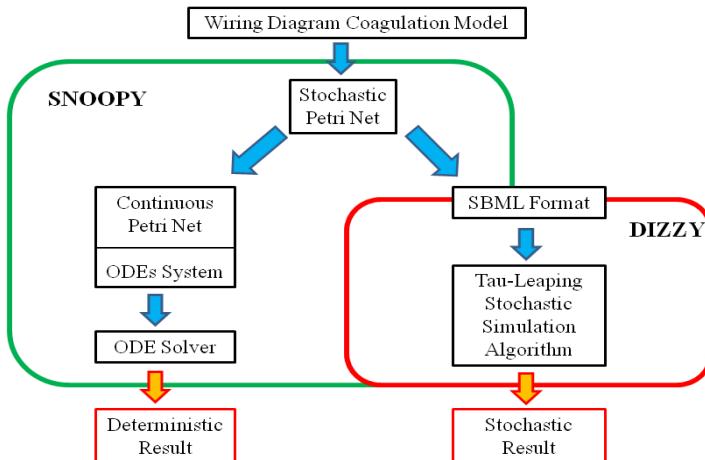


Fig. 5. Scheme of the simulation framework

5.2 Models Description

This work is focused on comparing the behaviour of a “healthy” subject with that of an “unhealthy” subject. We generate different models starting from the same Stochastic Petri Net shown in section 2, but changing the initial marking to represent the effect of a prothrombotic event. In particular, the unhealthy models have an higher initial number (from 10-fold to 1000-fold) of molecules in place representing the trigger factor of the extrinsic pathway, the Tissue Factor (TF place). This condition is clinically supported, reflecting the hypercoagulability state that arise locally during atherosclerotic plaque rupture (Reininger et al., 2010).

A second factor influencing the system is the VIIaTF complex which, as highlighted in literature (Monroe and Key, 2007), plays an important role in the thrombus formation process. A minimal amount is needed to start the coagulation cascade, but an excess of this complex due to prior cardiovascular inflammatory events can significantly affect the coagulation process. Therefore, we test the behaviour of our model with different amount of this factor, comparing a physiological amount with a pathological one, represented by a 2-fold and a 10-fold increase.

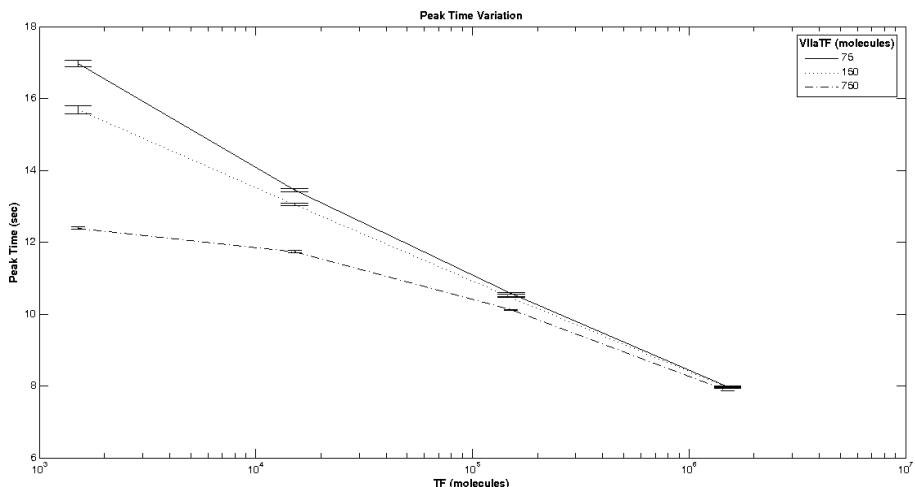


Fig. 6. Variation in mean and standard deviation of Thrombin (IIa) peak time

It is important to note that the reaction rate constants has not been modified from healthy to unhealthy models, because biological evidence proves that enzyme kinetics does not vary; nonetheless, the reaction firing rates change considerably between the two models because of the initial marking, promoting procoagulatory events.

Table 1. Mean and standard deviation of peak value and peak time, in healthy and unhealthy models

Model	VIIaTF Normal		VIIaTF 2-Fold		VIIaTF 10-Fold	
	Thrombin Mean Peak Time (sec)	Thrombin St.Dev. Peak Time (sec)	Thrombin Mean Peak Time (sec)	Thrombin St.Dev. Peak Time (sec)	Thrombin Mean Peak Time (sec)	Thrombin St.Dev. Peak Time (sec)
Healthy	16,96	0,106	15,67	0,091	12,38	0,037
Unhealthy (TF 10-fold)	13,44	0,051	13,04	0,044	11,73	0,026
Unhealthy (TF 100-fold)	10,56	0,017	10,46	0,017	10,11	0,015
Unhealthy(TF 1000-fold)	7,97	0,008	7,94	0,008	7,86	0,008

5.3 Computational Results

As we can see from Fig.7a, the amount of generated thrombin does not change from the healthy model to the unhealthy one, because the total availability of its precursor (prothrombin, II) is the same. Instead, we can see an anticipation of the growth in the unhealthy model, which is due only to the change of initial condition. This is observable even with a deterministic approach, but only from the stochastic results from in Table 1 we can notice how the unhealthy models show a lower intrapersonal variability compared to the healthy model, which shows an higher variability. A likely explanation of the lower variability in the unhealthy subject is given by the augmented amount of TF molecules, which leads to an higher rate for the procoagulant reactions. Since the rate of the inhibitory reactions is less affected, they will fire with a lower

frequency, thus reducing the noise on the main cascade. We can also see this effect just upstream of the activation of thrombin (Fig. 7b), e.g. in XaVaPL complex (prothrombinase), which is the main responsible of the generation of prothrombin. This is consistent with the clinical evidence, where the biological variability is lower in patients with pro-thrombotic conditions, and higher in patients with pro-haemorrhagic phenotypes. The same effect is produced by an increased amount of factor VIIaTF, which also limits the influence of the increment of TF amount (high levels of TF do not affect a system with high levels of VIIaTF).

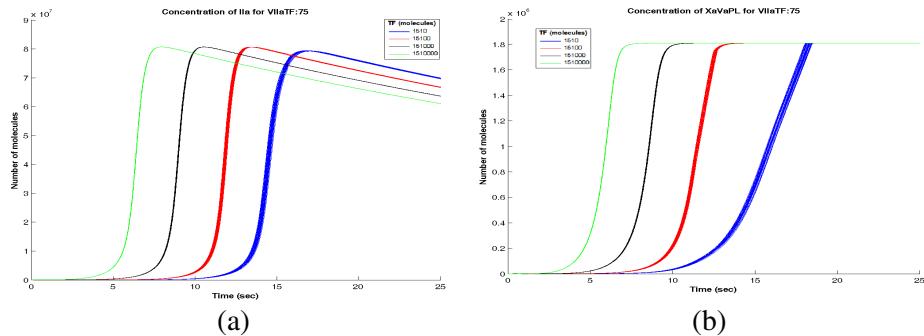


Fig. 7. (a) Thrombin (IIa) molecules trend in healthy and unhealthy models; (b) The same intra-variability seen in prothrombinase (XaVaPL).

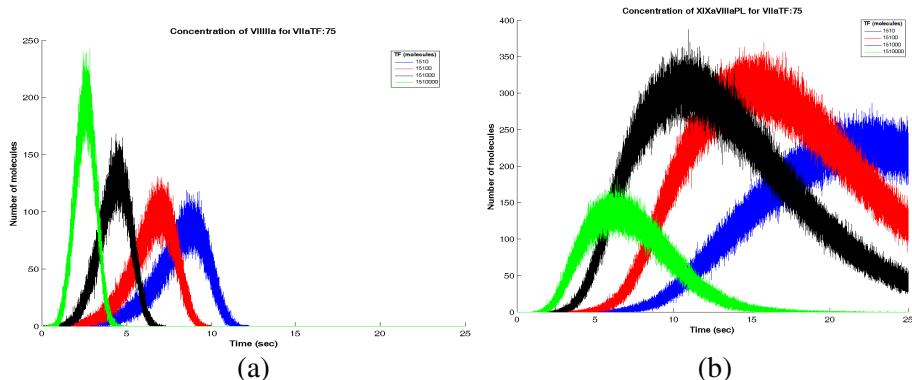


Fig. 8. Trend in molecular number for VIII-IIa (a) and XIXaVIIIaPL (b), which shows a strong fluctuability

Besides the different variability among models (Fig. 7), there is another aspect which is captured only by the stochastic approach. The number of molecules involved in many reactions (Fig. 8) shows strong fluctuability during critical phases of the simulation, when the system is at the peak of its activity, a feature which is impossible to detect using a deterministic approach, and very challenging even in a wet-lab experiment. Although being a preliminary analysis, this result can motivate further analysis on how much specific molecules react when the system is at the peak of its activity.

6 Conclusion

In this paper we presented a SPN model for the extrinsic coagulation, with the purpose of simulating the behavior of the whole system. We employed Gillespie's Tau-Leaping SSA in order to reduce the simulation time while retaining a good approximation, and we developed models describing "healthy" and "unhealthy" subjects.

We proved that a stochastic approach allows to detect important features that deterministic models cannot discover: first, there are molecule types which appears in a very low amount (up to 10 molecules), which only the stochastic method allows to simulate without mathematical artifacts; second, we shown that an increase in the quantity of Tissue Factor or VIIaTF complex reduces the degree of variation of the system, which is confirmed by clinical evidence

where the biological variability is lower in of patients with pro-thrombotic conditions, and higher in patients with pro-haemorrhagic phenotypes; finally, only the stochastic simulation identified the high fluctuability in the system during critical phases, which will allow to analyze deeper how much the particles reacts when the system is at the peak of its activity.

This model, which better capture the true variability of this complex system, represent the coagulation in a more realistic manner compared to the deterministic models appeared in literature. This approach sets the background for the analysis of more detailed models, which will include pharmacological interactions. Future improvements will also include further development of the simulation framework, including the automatic stochastic simulation of models with a wide range of parameters, and the performance analysis of additional simulation software, in order to compare their CPU time and memory requirements.

References

- Breitling, R., Gilbert, D., Heiner, M., Orton, R.: A structured approach for the engineering of biochemical network models, illustrated for signalling pathways. *Brief Bioinform.* 9(5), 404–421 (2008)
- Butenas, S., Mann, K.G.: Blood Coagulation. *Biochemistry* 57, 5–15 (2002)
- Butenas, S., van't Veer, C., Mann, K.G.: "Normal" Thrombin Generation. *Blood* 94(7), 2169–2178 (1999)
- Cao, Y., Gillespie, D.T., Petzold, L.R.: Efficient stepsize selection for the tau-leaping simulation method. *J. Chem. Phys.* 124, 044109 (2006)
- Cevenini, E., Bellavista, E., Tieri, P., Castellani, G., Lescai, F., Francesconi, M., Mishto, M., Santoro, A., Valensin, S., Salvioli, S., Capri, M., Zaikin, A., Monti, D., de Magalhaes, J.P., Franceschi, C.: Systems biology and longevity: an emerging approach to identify innovative anti-aging targets and strategies. *Curr. Pharm. Des.* 16(7), 802–813 (2010)
- Chatterjee, M.S., Denney, W.S., Jing, H., Diamond, S.L.: Systems Biology of Coagulation Initiation: Kinetics of Thrombin Generation in Resting and Activated Human Blood. *PLoS Comput. Biol.* 6(9), e1000950 (2010)
- Chu, A.J.: Tissue factor, blood coagulation, and beyond: an overview. *Int. J. Inflam.* 367284, 1–30 (2011)

- Corlan, A.D., John Ross, J.: Canalization effect in the coagulation cascade and the interindividual variability of oral anticoagulant response. A simulation study. *Theor. Biol. Med. Model.* 8, 37 (2011)
- Couris, R., Tataronis, G., McCloskey, W., Oertel, L., Dallal, G., Dwyer, J., Blumberg, J.B.: Dietary vitamin K variability affects International Normalized Ratio (INR) coagulation indices. *Int. J. Vitam. Nutr. Res.* 76(2), 65–74 (2006)
- Danforth, C.M., Orfeo, T., Mann, K.G., Brummel-Ziedins, K.E., Everse, S.J.: The impact of uncertainty in a blood coagulation model. *Math. Med. Biol.* 26(4), 323–336 (2009)
- Dittamo, C., Cangelosi, D.: Optimized Parallel Implementation of Gillespie's First Reaction Method on Graphics Processing Units. In: International Conference on Computer Modeling and Simulation, ICCMS 2009, pp. 156–161 (2009)
- Filipovic, N., Kojic, M., Tsuda, A.: Modelling thrombosis using dissipative particle dynamics method. *Philos. Transact. A Math. Phys. Eng. Sci.* 366(1879), 3265–3279 (2008)
- Gaffney, P.J., Edgell, T.A., Whitton, C.M.: The Haemostatic Balance – Astrup Revisited. *Haemostasis* 29, 58–71 (1999)
- Gillespie, D.T.: Exact Stochastic Simulation of Coupled Chemical Reactions. *The Journal of Physical Chemistry* 81(25), 2340–2361 (1977)
- Gillespie, D.T.: Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.* 115, 1716–1733 (2001)
- Goss, P.J.E., Peccoud, J.: Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. *Proc. Nat. Acad. Sci. USA* 95, 6750–6754 (1998)
- Heiner, M., Gilbert, D., Donaldson, R.: Petri Nets for Systems and Synthetic Biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 215–264. Springer, Heidelberg (2008)
- Khanin, M.A., Rakov, D.V., Kogan, A.E.: Mathematical model for the blood coagulation prothrombin time test. *Thromb. Res.* 89(5), 227–232 (1998)
- Li, H., Petzold, L.: Efficient Parallelization of the Stochastic Simulation Algorithm for Chemically Reacting Systems On the Graphics Processing Unit. *Int. J. of High Perf. Comp. Appl.* 24, 107–116 (2009, 2010)
- Lo, K., Denney, W.S., Diamond, S.L.: Stochastic modeling of blood coagulation initiation. *Pathophysiol. Haemost. Thromb.* 34(2-3), 80–90 (2005)
- Luan, D., Zai, M., Vamer, J.D.: Computationally Derived Points of Fragility of a Human Cascade Are Consistent with Current Therapeutic Strategies. *PLoS Comput. Biol.* 3(7), e142 (2007)
- Mackman, N.: Role of Tissue Factor in Hemostasis, Thrombosis, and Vascular Development. *Arterioscler. Thromb. Vasc. Biol.* 24, 1015–1022 (2004)
- Monroe, D.M., Key, N.S.: The tissue factor-factor VIIa complex: procoagulant activity, regulation, and multitasking. *J. Thromb. Haemost.* 5(6), 1097–1105 (2007) (review)
- Ramsey, S., Orrell, D., Bolouri, H.: Dizzy: stochastic simulation of large-scale genetic regulatory networks. *J. Bioinform. Comput. Biol.* 3, 415–436 (2005)
- Reininger, A.J., Bernlochner, I., Penz, S.M., Ravanat, C., Smethurst, P., Farndale, R.W., Gachet, C., Brandl, R., Siess, W.: A 2-Step Mechanism of Arterial Thrombus Formation Induced by Human Atherosclerotic Plaques. *J. Am. Coll. Cardiol.* 55, 1147–1158 (2010)
- Rohr, C., Marwan, W., Heiner, M.: Snoopy—a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics* 26(7), 974–975 (2010)
- Shaw, O., Steggles, J., Wipat, A.: Automatic Parameterization of Stochastic Petri Net Models of Biological Networks. *Electronic Notes in Theoretical Computer Science* 151, 111–129 (2006)

Srivastava, R., Peterson, M.S., Bentley, W.E.: Stochastic kinetic analysis of the Escherichia coli stress circuit using sigma(32)-targeted antisense. *Biotechnol. Bioeng.* 75(1), 120–129 (2001)

Steffel, J., Lüscher, T.F., Tanner, F.C.: Tissue Factor in Cardiovascular Diseases. Molecular Mechanisms and Clinical Implications. *Circulation* 113, 722–731 (2006)

Wajima, T., Isbister, G.K., Duffull, S.B.: A comprehensive model for the humoral coagulation network in humans. *Clin. Pharmacol. Ther.* 86, 290–298 (2009)

Processing the Biomedical Data on the Grid Using the UNICORE Workflow System

Marcelina Borcz^{1,2}, Rafał Kluszczyński², Katarzyna Skonieczna^{3,4},
Tomasz Grzybowski³, and Piotr Bała^{1,2}

¹ Faculty of Mathematics and Computer Science, Nicolaus Copernicus University,
Chopina 12/18, 87-100 Toruń, Poland

² Interdisciplinary Centre for Mathematical and Computational Modelling,
University of Warsaw, Pawiańskiego 5a, 02-106 Warsaw, Poland

³ Department of Molecular and Forensic Genetics, Institute of Forensic Medicine,
Ludwik Rydygier Collegium Medicum, Nicolaus Copernicus University,
Skłodowskiej-Curie 9, 85-094 Bydgoszcz, Poland

⁴ The Postgraduate School of Molecular Medicine, Medical University of Warsaw,
Żwirki i Wigury 61, 02-091 Warsaw, Poland

Abstract. The huge amount of the biological and biomedical data increases demand for significant disk space and computer power to store and process them. From its beginning the Grid has been considered as possibility to provide such resources for the life sciences community. In this paper authors focus on the UNICORE system which enables scientists to access Grid resources in a seamless and secure way. Authors have used the UNICORE middleware to automate experimental and computational procedure to determine the spectrum of mutations in mitochondrial genomes of normal and colorectal cancer cells. The computational and storage resources have been provided by Polish National Grid Infrastructure PL-Grid.

1 Introduction

The amount of data which biologists have to handle is usually too big to be stored and processed using desktop or even more powerful single computers in the laboratory. The scientists are forced to use clusters in supercomputer centers or even multiple resources combined into a single infrastructure. The problem is especially important for the processing biomolecular and genetic data necessary for diagnostics and research. A good example is analysis of the genomic data generated by high throughput experimental systems. With the increasing availability of such devices there is growing demand to process experimental data effectively and provide results in hours rather than days or weeks. Usually, processing of genetic data is performed on the dedicated hardware installed at the experimental facilities or hospital. With the introduction of new, low cost sequencing systems this model is no longer valid and there is strong demand to acquire external resources for storage and computations. The obvious choice are grids and clouds since they can provide required infrastructure. Because security of the data is here a concern, the grid solutions with the strong security model based on the X509 certificates is natural solution.

Since there is number of the grid infrastructures around, the user can pick the most suitable according to the different metrics including cost. For the academic users, the natural choice is to use national grid infrastructures organized within the EGI framework. The PL-Grid - Polish Grid Infrastructure is a good example. It is available to the research and academic community free of charge and provides access through gLite and UNICORE middlewares.

UNICORE middleware [18] is one of the successful solutions used to build distributed infrastructure. Along with the UNICORE Rich Client (URC) it enables users to run programs, manage files and data transfers. It allows also to design and run workflows in a graphical environment which is easy to use. In this paper we present example solution which allows for the uniform access to the experimental data, their handling and analysis in the distributed environment which significantly reduces processing time.

The paper is organized as follows: the experiment is described in the first section of the paper. The second part focuses on the description of the UNICORE middleware and the UNICORE workflow system. It is followed by the specification of our solution and the experimental setup. The paper concludes with the summary and directions for further work.

2 Experiment

Recently an increasing number of studies have indicated that changes in mitochondrial genome sequence may contribute cancer phenotype [15,12]. Despite of that fact, the knowledge of mitochondrial DNA (mtDNA) variability in colorectal cancer is still limited, and until now the reliable spectrum of mtDNA somatic mutations has not been resolved [17]. Thus it is important to determine mtDNA variability in normal and tumor cells. Due to the usage of high-throughput GS FLX Instrument (Roche Diagnostics) up to 1 million reads (that correspond to the 1 million DNA fragments) of approximately 500 base pairs long could be generated independently in a single experiment [1]. The aforementioned ultra-deep DNA sequencing technology may aid the identification of low-level heteroplasmic, nucleotide variants. Therefore, we have used 454 sequencing technology to resolve mutational spectrum of the entire mitochondrial genome sequences of non-tumor and colorectal cancer cells. The particular aims of our research were:

- determination of the 18 complete mitochondrial genome sequences of tumor and matched non-tumor tissues obtained from 9 patients diagnosed with colorectal cancer,
- mtDNA sequences comparison with the reference sequence,
- mtDNA mutation identification,
- ultra high speed processing of mtDNA sequence data.

Mitochondrial DNA sequences were determined with 454 sequencing technology. In brief, mtDNA molecules were amplified in two overlapping fragments of about 8500 base pairs in length [10]. Next, the amplicones were nebulized into 400–500 base pairs fragments long, clonally amplified in emPCR and sequenced with GS FLX Instrument

(Roche Diagnostics). During the sequencing reaction digital images were subsequently captured by the CCD camera. The raw image data from a single experiment (approximately 30GB in size) were next converted with the use of GS Run Processor (Roche Diagnostics) into base-called results. Reports of the base-calling analysis were generated with GS Reporter (Roche Diagnostics). Thereafter obtained reads were aligned to a revised Cambridge Reference Sequence (rCRS) [3] with the use of a GS Reference Mapper (Roche Diagnostics), which enables mtDNA mutations to be detected. Roche provides software for the RedHat Linux free of charge to the equipment. All components use popular and standardized file formats (eg. fasta, ece) which makes it easy to interoperate.

3 Data Storage and Data Processing Infrastructure

The processing of the sequenced data requires significant computational resources not available as part of the typical experimental setup. We have decided to integrate GS FLX Instrument with the PL-Grid distributed infrastructure [13] as the data storage and processing system.

3.1 PL-Grid Infrastructure

The Polish Grid Infrastructure provides the Polish scientific community with an IT platform based on computer clusters, enabling research in various domains of e-Science. PL-Grid infrastructure enables scientists carrying out scientific research based on the simulations and large-scale calculations using the computing clusters as well as provides convenient access to distributed computing resources. The infrastructure supports scientific investigations by integrating experimental data and results of advanced computer simulations carried out by geographically distributed research teams. Polish Grid Infrastructure is a part of a pan-European infrastructure built in the framework of the EGI (European Grid Initiative), which aims to integrate the national Grid infrastructures into a single, sustainable, production infrastructure. PL-Grid infrastructure is both compatible and interoperable with existing European and worldwide Grid frameworks.

UNICORE is one of the middlewares offered to the users which enables access in a seamless and secure way. The UNICORE PL-Grid infrastructure is based on the resources provided by the main computational centers in Poland. It is built of individual sites which are described by the list of provided hardware, services and installed applications (see. Figure 1).

The PL-Grid portal provides users with the forms to perform registration and application for resources. It also delivers interface for the users to obtain self-generated certificates necessary to access the grid. The process itself uses an additional module registering users in the virtual organization. As a part of the user's registration there is performed data replication to VOMS and UVOS server depends on the middleware the user want to use. The portal allows to register any certificate issued by CA approved by EUGridPMA organisation [8] or additional Simple CA certificates used only internally in PL-Grid.

This solution allows to hide from the user less intuitive parts of the certificate issuing process and facilitates placement in the UVOS additional information in the form of groups or attributes. These informations are necessary to grant users with the proper privileges.

3.2 UNICORE

UNICORE is a system which provides an easy and secure access to the Grid resources. The system has been developed since 1997 [18] and has been successfully used in many European scientific projects contributing significantly to the increase of the popularity and applications of distributed computing. UNICORE is part of the European Middleware Initiative [9] where, together with gLite and ARC middlewares, developers continue their contribution to the Grid standards by increasing interoperability, manageability, usability and efficiency of the Grid services. The UNICORE system is also used by the National Grid Initiative infrastructures such as PL-Grid which provides resources to the scientific communities.

One of the main advantages of the UNICORE system is set of the UNICORE clients. There are three types of them: UNICORE Rich Client (URC), UNICORE Command-line Client (UCC) and High Level API (HiLA). UNICORE Rich Client enables users to design and run workflows in a graphical way which does not require knowledge of particular language used to define and store workflows. This capability is important for the end users since most scientific experiments have the workflow structure: the output of one step of the experiment is the input for another one. The URC allows for easy, graphical creation of the workflows and for the management of their execution. In the Grid context, workflow is the automation of the processes which involves the orchestration of a set of Grid services or agents in order to solve a problem or to define a new service [20]. UNICORE workflow system implements loops and if-else statements so the user can design sophisticated scientific experiments and run them just by pressing the run button. Once designed, the workflow can be saved and used many times with different data and parameters. All files created in the workflow can be managed in a graphical way in the client. Two-layered structure of the UNICORE workflow system allows to plug-in domain-specific workflow languages, workflow engines and various brokering strategies.

The UNICORE server side components fully adopt grid services paradigm. The computational systems are called Target Systems, and are built of two main UNICORE services: UNICORE/X and TSI which enable access to computing and data resources. All communication between the user and grid services is streamed through the UNICORE Gateway service to ensure secure access. In the Polish National Grid Infrastructure main UNICORE services are located at the Interdisciplinary Centre for Mathematical and Computational Modelling (ICM) at the University of Warsaw. ICM provides central services such as the registry of all sites' endpoints and Unicore Virtual Organization Service acting as a users management service. The UVOS is populated with the user data received from the PL-Grid portal which contains information about users, their privileges and certificates. This data is synchronized with the UVOS and is used to authorize and authenticate users.

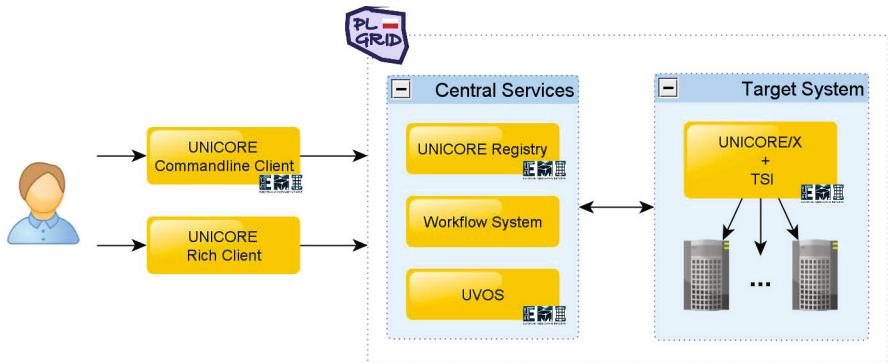


Fig. 1. Diagram of UNICORE infrastructure setup deployed in PL-Grid

UNICORE infrastructure in PL-Grid is built based on the European Middleware Initiative (EMI) releases, which are distributed from EMI repository as RPM packages [7] which allows for easy installation and configuration with only few additional dependencies.

3.3 Storage

PL-Grid infrastructure allows users to organize themselves in scientific groups (called teams), which can apply for common allocations, share computational resources and data. In the case of data, members of a group have access to special storage at every site enabling them to share data. UNICORE provides access to such storage by a simple configuration of Target System. A default access privileges (unix umask) can be set to ensure that newly created folders and files are accessible by default to all of the group members.

A typical PL-Grid target system running UNICORE middleware offers three types of storages:

- **user space** (also known as file-space), which contains files created for running tasks, it is usually fast filesystem (like Lustre [11]);
- **global storage**, which contains files not related with particular task, it is usually slower, but more reliable than user space;
- **group storage**, which contains files that can be accessible by all of the group members, it can be slow but is also reliable.

The PL-Grid infrastructure offers access to the group storage also from other grid middlewares enabling users to use different tools for data access.

3.4 UNICORE FTP

The key issue for integration of the experimental setup with the storage infrastructure is efficient data transfer. UNICORE/X offers three built-in protocols: BFT, RBYTEIO

and SBYTEIO. In the case of data upload, all of them transfer data through the UNICORE Gateway which introduces communication overhead resulting in low transfer speed. The solution is a new protocol called UNICORE File Transfer Protocol (UFTP) which has been recently implemented [16]. In order to transfer data with UFTP an additional UFTPD service has to be installed next to the UNICORE/X. The data transfer is therefore performed directly between UFTPD and UNICORE client. Data transfer uses dynamic firewall port opening mechanism by using passive FTP connections which allows to bypass the UNICORE Gateway. This solution is considered to be more secure than statically opened port ranges usually used for example by the GridFTP server [2].

In order to make UFTP protocol more secure, all file transfers are always initiated by the UNICORE/X server. To start file transfer client contacts UNICORE/X, determines the parameters of the transfer and then connects to the UFTPD to perform the actual transfer. The UNICORE/X passes client IP and transfer parameters to the UFTPD service which waits for connections for defined period of time. Communication between UNICORE/X and UFTPD is done through a secure “command port” accessible only by defined UNICORE/X servers which prevents non authorized usage. Another advantage of the UFTP protocol is ability to establish multiple parallel TCP connections per data transfer which significantly speeds up transfer.

4 Automation of the Experiment

The applications used as parts of the workflow developed for the processing of the genetics data work under the Roche License. Key components are parallelized using Message Passing Interface. The programs from the FLX suite are compiled for the Red Hat Linux and are installed by the site administrators on the PL-Grid target systems. Therefore it is important to make sure that only privileged users have permission to run it. This can be done using the group management available in the PL-Grid. Groups are mapped into the operating system’s groups. In this way the access to the installed applications is granted only for the proper team members for whom the license has been issued.

The developed workflow consists of three programs from the FLX program set. First, the GS Run Processor processes raw images generated by the FLX Instrument (Roche diagnostics). This data, which can exceed tens of gigabytes, is put into the UNICORE storage in an automatic way. After a new file or directory with data is created as a result of experiment, the UNICORE Commandline Client program is used to put automatically data to the PL-Grid UNICORE target system storage (see Figure 2). Because of the size of the data a UFTP protocol is used for transfers. Appropriate access rights to the group storage ensure privacy.

The GS Run Processor has several components. We run the `runAnalysisPipe` script for full processing of the acquired data and use the `GS_LAUNCH_MODE` environmental variable to set MPI mode enabling us to use multiple worker nodes on the target system. The next part of the workflow can be run simultaneously on the available resources. The GS Reporter generates all reports files from CWF files created in the

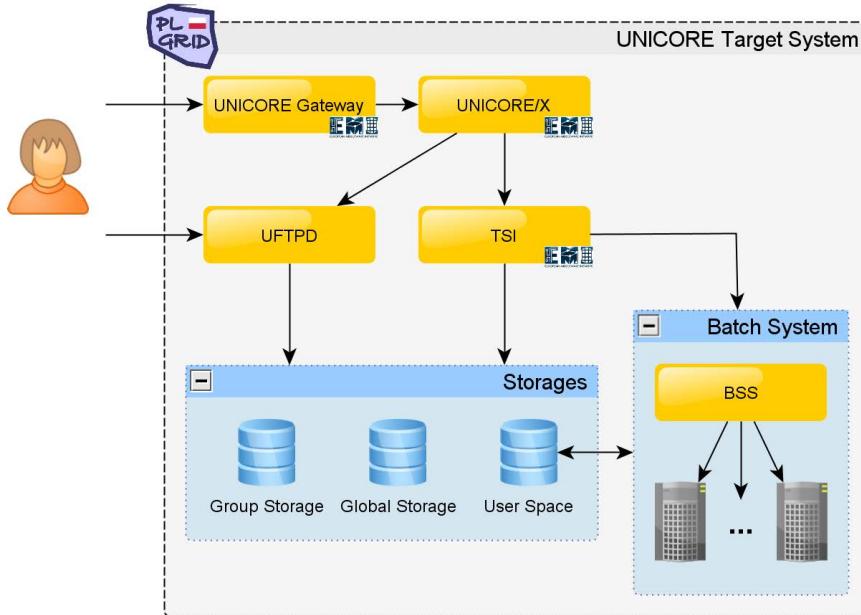


Fig. 2. The detailed view of the UNICORE target system with the different types of storage

previous job. The GS Reference Mapper consists of two steps which create the mapping project and align the reads against a reference sequence. The example workflow edited in the UNICORE Rich Client is shown in the Figure 3.

The execution of the workflow ends when all workflow components are finished. The workflow branches are run in parallel on different target systems to speed up execution. Each workflow component represent application which can be run in parallel using MPI on the target system. The workflow can be stored on the disk and then loaded to the URC. The UNICORE uses its own format for internal representation of the workflows which is translated to the single work assignments. The significant advantage of the UNICORE Rich Client is automatic generation of the necessary files based on the graphical representation of the workflow. Therefore the knowledge of particular workflow definition language is not required and workflow creation and modifications are performed using intuitive GUI.

It is up to the user to decide where the output of each workflow step is stored. Files can be directed to the global or group storage or simply stored in the directory of the task. In the context of workflows the second option is better since it usually takes less time to copy the data between jobs.

It should be mention that a user has access to all files generated by each job despite their physical location. Data can be downloaded to the local computer or permanently stored in the distributed storage. The workflow can be extended to include additional programs. For example Blast [4], Clustal [6] or R [14] applications can be used for further processing.

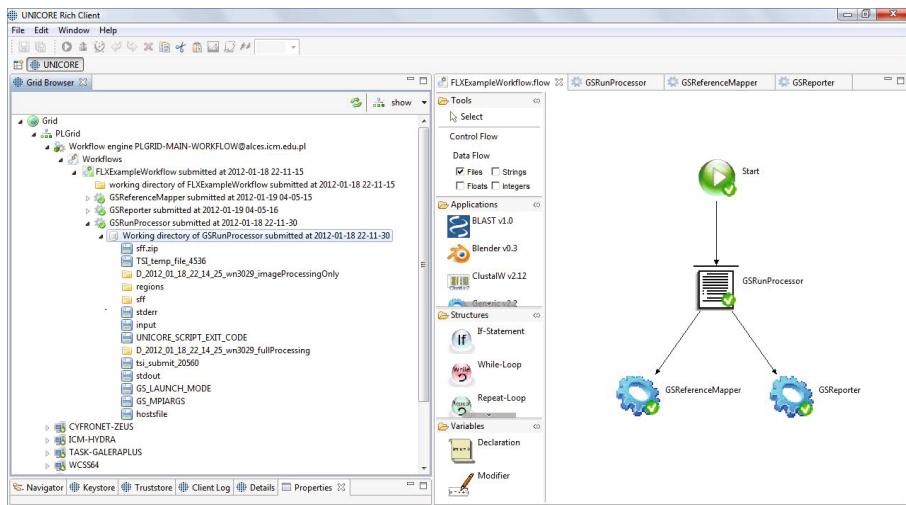


Fig. 3. The UNICORE Rich Client view on the designed workflow to process acquired biomedical data

5 Results

The described setup has been used to store and to process experimental data on the PL-Grid infrastructure. Single sequencing experiment generated ca. 30 GB of data which is organized in the 834 files of the size of about 33 MB each. The transfer of the individual file from the workstation attached to the sequencer takes few second, the overall time to transfer experiment's results was 3-4 hours. One should note that this process has been fully automated and did not require user assistance. Once data is stored on the Grid it can be easily used as the input for analysis.

Table 1. Performance results of sequence analysis for various hardware. The different hardware systems are available to the user through UNICORE target systems provided by the PL-Grid.

Processor type	Cache size	Interconnect	Number of cores	Time (hours)
Intel Xeon CPU @ 3.60GHz	1024 kB	none	1	70.0
AMD Interlagos Opteron Processor 6272 @ 2.10GHz	2048 kB	Gigabit	64 (1 x 64 cores)	6.5
AMD Opteron Processor 6174 @ 2.20GHz	512 kB	Gigabit	64 (8 x 8 cores) 96 (8 x 12 cores)	4.5 4.5
Intel Xeon CPU, X5660 @ 2.80GHz	12288 kB	Infiniband	64 (8 x 8 cores) 96 (8 x 12 cores)	2.5 2.5

The important benefit of the UNICORE workflow system is that workflow once developed can be run on the different target systems. Therefore we were able to run the analysis workflow on the available hardware using parallel execution. The performance of the GS Run Processor, the most time consuming step, presented in the Table 1 shows significant speedup for 64 cores used compared to the single core run. Further increase of the number of cores has no effect on execution time. One should note significant difference in the execution time between different processors. This is caused by the different size of the internal cache which is the largest for the Xeon processor. UNICORE allows user for easy pick up optimal one.

6 Conclusions

The usage of the storage and processing power of the distributed infrastructures allowed for significant reduction of the analysis time of the results of mitochondrial genome sequence runs which was the bottleneck in the process. The time required for the data processing needed for the final analysis has been reduced to hours instead of several days. Available distributed storage allowed for tracking multiple data which opens up field for detailed statistical analysis. The data processing has been automated and simplified based on the UNICORE workflows. In practice all stages of the data processing are run automatically reducing manual work and unnecessary delays. Practically unlimited storage and fast and reliable data transfer mechanisms allow to offer users flexible and easy to use storage accessible simply on the Internet. This functionality is especially attractive to biological and medical users who can focus on their research instead of mastering computer science details necessary to process data.

7 Future Work

Current set up for high-throughput GS FLX Instrument is a starting point for building dedicated infrastructure to process biomedical data. We plan to significantly increase data processing functionality by adding new applications and by creation more complicated workflows which will automate analysis. We also plan to provide more flexibility to biologists and medicians using the Gridbeans for gsMapper and other applications used. A GridBean is a plugin for UNICORE Rich Client which provides graphical interface to an application and allows for easy change of the parameters used for execution. Thanks to this users will be able to modify workflow execution without assistance of the IT staff and thus will receive further flexibility in running complicated data analysis.

Acknowledgments. This research was supported in part by the PL-Grid Infrastructure and by the EMI project (UE RI-261611). The study was supported by the Ministry of Science and Higher Education (grant no.: NN 301 075 839).

References

1. 454 sequencing technology website, <http://www.454.com>
2. Allcock, W.: GridFTP: Protocol Extensions to FTP for the Grid. Global Grid ForumGFD-R-P020 (2003)
3. Andrews, R.M., Kubacka, I., Chinnery, P.F., Lightowlers, R.N., Turnbull, D.M., Howell, N.: Reanalysis and revision of the Cambridge reference sequence for human mitochondrial DNA. *Nat. Genet.* 23, 147 (1999)
4. Borc, M., Kluszczyński, R., Bała, P.: BLAST Application on the GPE/UnicoreGS Grid. In: Lehner, W., Meyer, N., Streit, A., Stewart, C. (eds.) Euro-Par 2006 Workshops. LNCS, vol. 4375, pp. 245–253. Springer, Heidelberg (2007)
5. Benedyczak, K., Stolarek, M., Rowicki, R., Kluszczyński, R., Borcz, M., Marczak, G., Filocha, M., Bała, P.: Seamless Access to the PL-Grid e-Infrastructure Using UNICORE Middleware. In: Bubak, M., Szepieniec, T., Wiatr, K. (eds.) PL-Grid 2011. LNCS, vol. 7136, pp. 56–72. Springer, Heidelberg (2012)
6. Clustal software website, <http://www.clustal.org>
7. EMI Software Repository, <http://emisoft.web.cern.ch/emisoft>
8. EUGridPMA organisation website, <http://www.eugridpma.org>
9. European Middleware Initiative (EMI) website, <http://www.eu-emi.eu>
10. Fendt, L., Zimmermann, B., Daniaux, M., Parson, W.: Sequencing strategy for the whole mitochondrial genome resulting in high quality sequences. *BMC Genomics* 10, 139 (2009)
11. Lustre product website, <http://www.lustre.org>
12. Petros, J.A., Baumann, A.K., Ruiz-Pesini, E., et al.: mtDNA mutations increase tumorigenicity in prostate cancer. *Proc. Natl. Acad. Sci. U.S.A.* 102, 719–724 (2005)
13. PL-Grid project website, <http://www.plgrid.pl>
14. R environment website, <http://www.r-project.org>
15. Sánchez-Aragó, M., Chamorro, M., Cuevva, J.M.: Selection of cancer cells with repressed mitochondria triggers colon cancer progression. *Carcinogenesis* 31, 567–576 (2010)
16. Schuller, B., Pohlmann, T.: UFTP: High-Performance Data Transfer for UNICORE. In: Romberg, M., Bala, P., Müller-Pfefferkorn, R., Mallmann, D. (eds.) Proceedings of UNICORE Summit 2011, Forschungszentrums Jülich. IAS Series, vol. 9, pp. 135–142 (2011) ISBN 978-3-89336-750-4
17. Skonieczna, K., Malyarchuk, B.A., Grzybowski, T.: The landscape of mitochondrial DNA variation in human colorectal cancer on the background of phylogenetic knowledge. *Biochim. Biophys. Acta* 1825, 153–159 (2011)
18. UNICORE middleware website, <http://unicore.eu>
19. UVOS project website, <http://uvos.chemomentum.org>
20. Yu, J., Buyya, R.: A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record* 34(3), 44–49 (2005)

Multicore and Cloud-Based Solutions for Genomic Variant Analysis

Cristina Y. González¹, Marta Bleda^{1,2}, Francisco Salavert^{1,2}, Rubén Sánchez¹, Joaquín Dopazo^{1,2,3}, and Ignacio Medina^{1,3}

¹ Computational Genomics Institute,

Centro de Investigación Príncipe Felipe (CIPF), Valencia, Spain

² CIBER de Enfermedades Raras (CIBERER), Valencia, Spain

³ Functional Genomics Node, Instituto Nacional de Bioinformática (INB) at CIPF,
Valencia, Spain

{cgonzalez,mbleda,fsalavert,rsanchez,jdopazo,imedina}@cipf.es

Abstract. Genomic variant analysis is a complex process that allows to find and study genome mutations. For this purpose, analysis and tests from both biological and statistical points of view must be conducted. Biological data for this kind of analysis are typically stored according to the Variant Call Format (VCF), in gigabytes-sized files that cannot be efficiently processed using conventional software.

In this paper, we introduce part of the High Performance Genomics (HPG) project, whose goal is to develop a collection of efficient and open-source software applications for the genomics area. The paper is mainly focused on HPG Variant, a suite that allows to get the effect of mutations and to conduct genomic-wide and family-based analysis, using a multi-tier architecture based on CellBase Database and a RESTful web service API. Two user clients are also provided: an HTML5 web client and a command-line interface, both using a back-end parallelized using OpenMP. Along with HPG Variant, a library for VCF files handling and a collection of utilities for VCF files preprocessing have been developed.

Positive performance results are shown in comparison with other applications such as PLINK, GenABEL, SNPTEST or VCFtools.

Keywords: Multicore, OpenMP, web service, genomic variant analysis, mutation.

1 Introduction

The application of Next Generation Sequencing (NGS) technologies is uncovering an unexpected amount of genomic variability. Detecting causal mutations related with particular disorders has become a big challenge given that most variants in patients are likely to be neutral [1]. Thus, genomic variant analysis requires an exhaustive cleaning of the generated data together with the application of several analysis and statistical tests.

Biological data for this kind of analysis are typically stored in files according to the Variant Call Format (VCF) [2], a plain text format with very concrete

information about each variant, such as: the chromosome and position where it has occurred, the bases in the reference genome and sequenced samples, and diverse information about the quality, coverage and other parameters of the reading. The size of these files varies from some megabytes to several gigabytes, in some cases impossible to be processed using standard workstations. In the near future, their size will probably increase in an order of magnitude, making this situation even more critical.

The goal of the High Performance Genomics project [3] is to develop a collection of efficient and open-source software applications for the genomic-scale data analysis. This paper is focused on HPG Variant, a suite that allows to get the effect of mutations via a RESTful web service architecture, and to conduct genomic-wide and family-based variant analysis. It avoids some limitations present in other available tools in relation to: (i) time, taking advantage of the parallel capabilities of current hardware in order to improve performance and (ii) memory usage, managing memory carefully in order to allow to process files of arbitrary size.

Along with the HPG Variant suite, a library for VCF files handling was developed. It provides an API also used from HPG VCF Tools, a collection of utilities for VCF file preprocessing. Both the library and the utilities are briefly described.

In order to check the suitability of these applications, some benchmarks have been conducted in comparison to others such as VCFtools [2], PLINK [4,5], GenABEL [6] and SNPTEST [7].

2 HPG Variant

HPG Variant is a suite of tools for performing tests and analysis of genomic variants. It currently consists of two main parts, a variant effect prediction tool named *effect*, and another one oriented to genome-wide association analysis (GWAS) named *gwas*.

Effect is, as its name points, a tool for predicting the effect or consequence type of variants. The effect of a genomic variant measures the putative biological consequence of a mutation, i.e. mutations can be located outside or inside genes or functional DNA regions, thus having different biological consequences. HPG Variant *effect* ports the existing VARIANT tool [8] to a more efficient implementation, and consists of a system with multiple layers, the lowest being CellBase DB [9], which stores information about variants and their consequence types for many species. This database can be queried through a complete set of RESTful web services using multiple user clients.

GWAS includes parallel implementations of several genomic analysis. During the past years, the application of these analysis in DNA microarray-based studies has been determinant in the identification of causal variants [10]. Unfortunately, the new sequencing technology is producing such amount of data that it cannot be easily analyzed using traditional programs. HPG Variant *gwas* tool takes as

a reference some tests available in PLINK, improving their performance both in execution time and memory usage, as well as using standard file formats in the Next Generation Sequencing context. The implemented features include genomic-wide association analysis such as chi-square and Fisher's exact test, and family-based analysis such as transmission disequilibrium test (TDT). Mendelian errors are automatically removed during the analysis.

The following sections introduce both the architecture and the user interface of HPG Variant.

2.1 Architecture

HPG Variant consists of a 3-layered architecture with a C back-end, a RESTful web services API and a biological database known as CellBase. The architecture design of the last two has been specially focused on fault-tolerance, thus providing a high-availability solution with no single point of failure. The architecture is shown from a global perspective in Fig. 1.

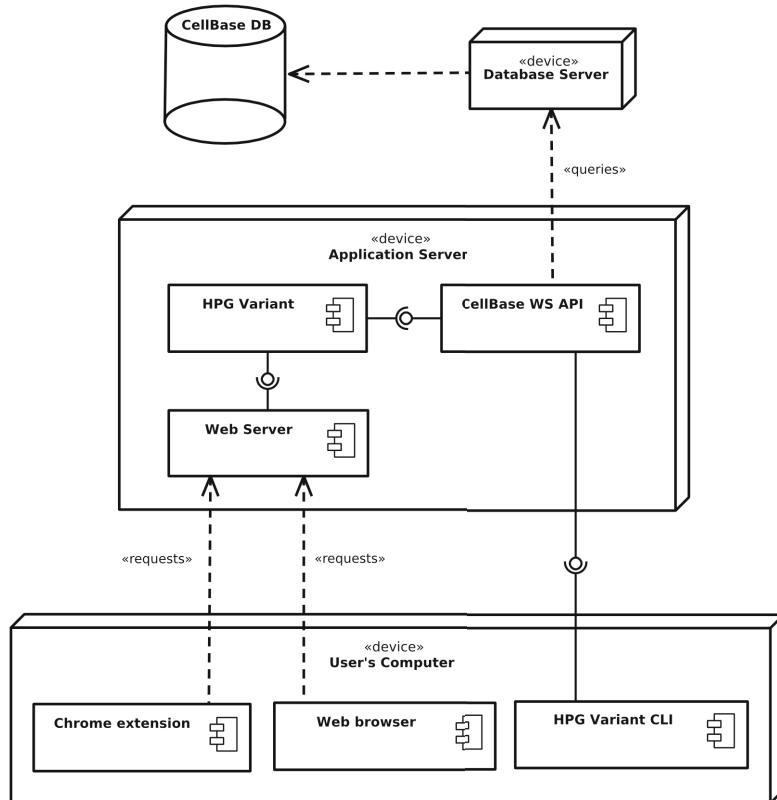


Fig. 1. HPG Variant *effect* architecture

CellBase, hosted and maintained at Príncipe Felipe Research Center database servers, stores more than 200 GB of data about multiple species, extracted from several biological databases and sources. Database optimizations allow not to spend more than 5 milliseconds on any query. In order to provide a high availability and load balancing, it has been implemented using a MySQL replication cluster, with Keepalived¹ and HAProxy² services configured. For the sake of simplicity, in Fig. 1 it has been represented as one pair server-database.

CellBase is not remotely accessed via direct queries but through specific RESTful API calls [11] that composes those queries. This provides a homogeneous yet still efficient way of retrieving information. The web services layer has been implemented in Java using the Java Jersey library³ and the database is locally accessed using Hibernate⁴.

The back-end is implemented according to both task and data parallelization schemes. Using OpenMP sections, it executes 3 main threads in a double producer-consumer. The application flow is shown in Fig. 2, where each “alt” section corresponds to one of these threads. These “alt” sections apply to all the HPG Variant tools, but the inner “worker threads” section is specific to HPG Variant *effect*.

The first thread is responsible of reading the VCF file. One of the main problems faced during the implementation of the back-end was that VCF files can be several gigabytes-sized and impossible to manage in a single iteration. For this reason, reading was implemented in several steps, each one retrieving a certain amount of lines from the file and queueing them into a list shared between threads. This list controls memory usage by pausing insertions when the maximum authorized size is reached, and resuming when at least one position is free. Optionally, the thread here described cannot only read but also parse the file and generate the corresponding data structures.

The second thread consumes each bunch of lines and forks a set of worker threads in order to process them. The results are also queued in another shared list. Depending on the workload of the task to be performed and in order to maximize the overlapping time between I/O and computation routines, the worker threads can assume the responsibility of parsing the input text and creating the data structures. For example, the *effect* tool involves several network connections for querying CellBase, which are more expensive than text parsing; in this case, parsing is managed by the first thread. On the other hand, tests such as TDT or basic case-control association involve very simple computations. Therefore, the worker threads both create the data structures and compute the results. Another configurable argument that also allows to reduce the time that worker threads must wait is the number of lines in a bunch. Most of the application arguments related to parallelism can be tweaked in order to improve performance. At the moment, their value must be set via command-line or a configuration file, although possible ways of automation are being studied.

¹ Keepalived project website: <http://www.keepalived.org>

² HAProxy project website: <http://haproxy.1wt.eu>

³ Jersey library website: <http://jersey.java.net>

⁴ Hibernate project website: <http://www.hibernate.org/>

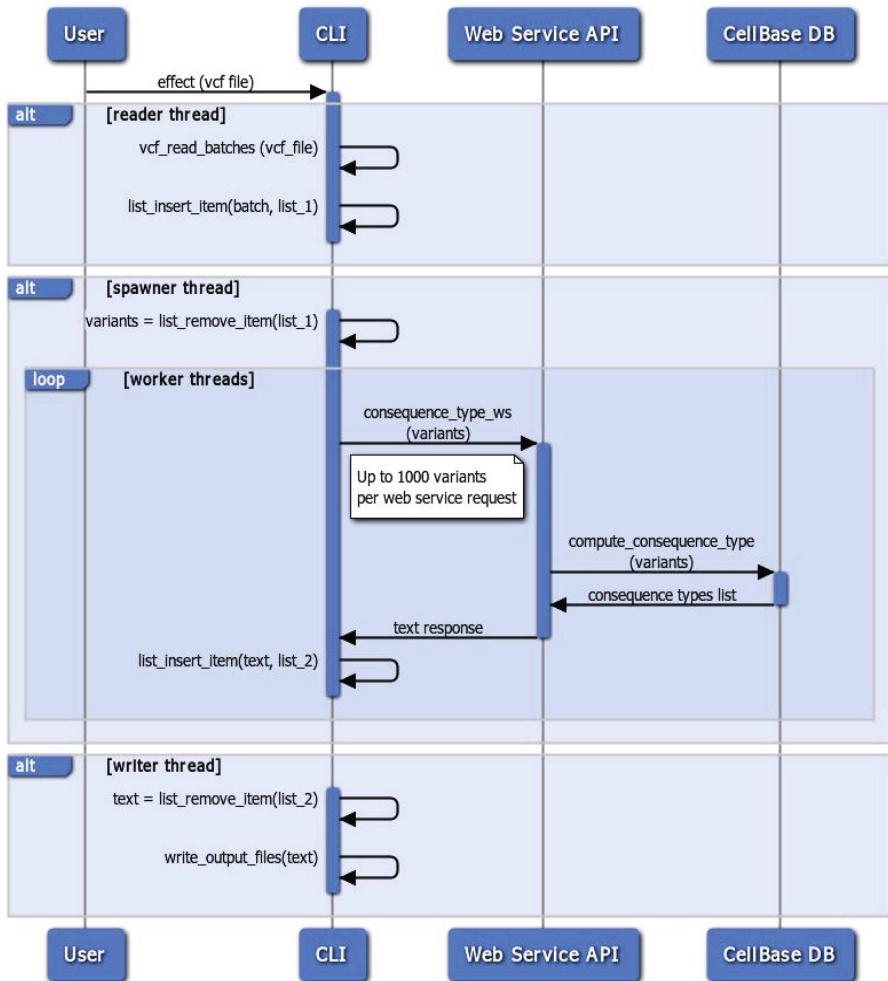


Fig. 2. HPG Variant *effect* sequence diagram

Finally, the third main thread consumes the list populated by the second one and writes the output to one file, as in statistical tests, or one per consequence type, as in the *effect* tool.

Additionally, a web application has been developed following the Software as a Service (SaaS) paradigm. Users can upload and permanently store their VCF data files in the machines we have deployed, conduct their studies, and visualize and download the results. It can be accessed at the URL <http://variant.bioinfo.cipf.es>. At the moment, only the *effect* tool is available this way, but *gwas* and HPG VCF Tools will also be accessible, providing the users with a full online solution. The web application will be described with more detail in

Sect. 2.2. The back-end can also be downloaded and run as a command-line standalone application, in case the user wants to avoid uploading huge files.

2.2 Interface

As stated at the beginning of Sect. 2, HPG Variant includes a collection of tools accessible via different user interfaces. This section describes the command-line interface in detail and how the web interface interacts with it.

The HPG Variant command-line is a combination of the name of the tool to invoke, a list of arguments available across the whole application, and another list of arguments specific to the tool.

Listing 1.1. Invoking hpg-variant

```
hpg-variant effect --vcf-file input1.vcf --region 1:10000-20000 --quality
40 --num-threads 4 --outdir path/to/output/folder
```

In Listing 1.1, the *effect* tool receives as input the file *input1.vcf*, which is filtered by a certain genomic region and a minimum quality. Four threads are used when parallelism is available and the results are written to the folder set in the *outdir* argument. When invoking the *gwas* tool, the test or analysis to conduct must be also specified as an argument, such as *--tdt* or *--fisher*.

All the tools in HPG Variant are able to filter their input. Some of the criteria available are region, quality and number of alleles, among others. These filters will be described in more detail in Sect. 4. In addition, other arguments used from the whole application are related to input and output, parallelization and memory management.

The application output varies for each tool. *Effect* generates a set of text files grouped by the consequence type of the variants, another one with all the consequence types found and a summary file with the count of each one. For a detailed explanation of the output see http://docs.bioinfo.cipf.es/projects/variant/wiki/Output_columns.

TDT analysis output is very similar to PLINK's. It shows the chromosome and position where the variant occurred, the possible alleles and the times they are transmitted, the odds ratio, the chi-square value and the p-value. Output of other tools has been suppressed for brevity.

Listing 1.2. GWAS TDT output

CHR	BP	A1	A2	T	U	OR	CHISQ	P-VALUE
1	742429	C	T	11	9	1.222222	0.200000	0.654721
1	767376	A	G	0	1	0.000000	1.000000	0.317311
1	769185	G	A	11	5	2.200000	2.250000	0.133614

Regarding the web application, it simplifies the use of the CLI by automatically generating and queuing the whole *effect* command. Moreover, it shows the

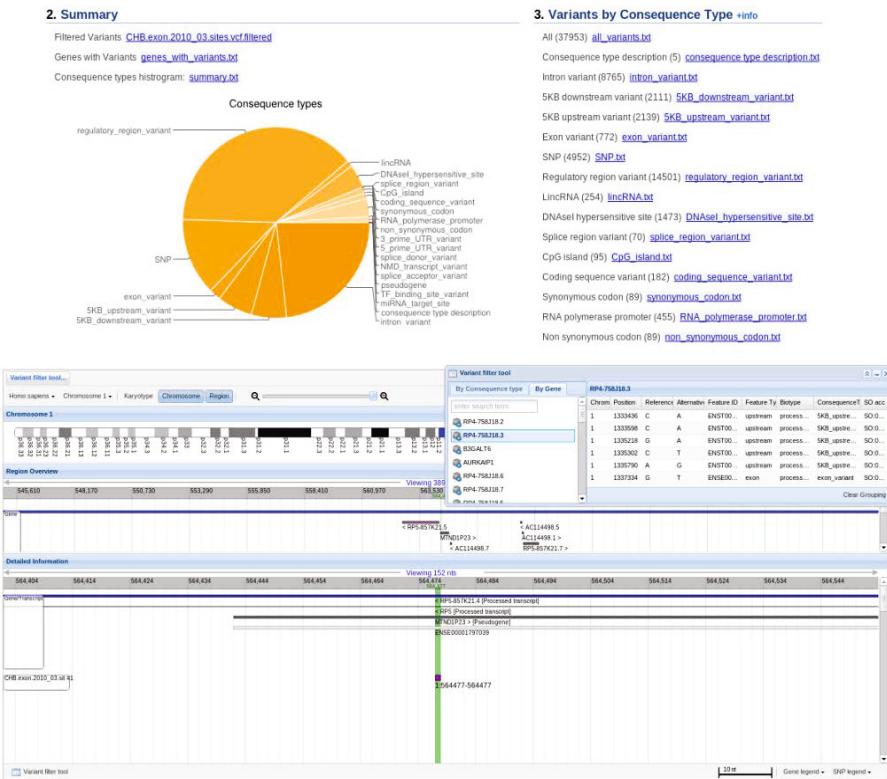


Fig. 3. Some results of HPG Variant effect shown in its website

results in a more understandable way: (i) the summary section includes the analyzed VCF file (or a filtered version if applicable), a file with the genes that contain variants and a pie chart showing the ratio of occurrence of each consequence type, (ii) in another section users will find the consequence type files cited above and (iii) an embedded instance of the Genome Maps [12] browser allows to explore the genome along with the variants analysed (see Fig. 3). Also, results are kept in our servers and users can revisit or share them from any other computer.

3 HPG VCF Tools

HPG VCF Tools is a set of tools for preprocessing, filtering and manipulating of VCF files. The implementation aims to avoid excessive time consumption in tedious preprocessing tasks. Existing solutions are slow and consume a lot of memory due to their implementation in high-level languages like Perl.

At the moment, HPG VCF Tools includes parallel implementations for filtering, splitting and getting statistics. Merging will be included in the short-term.

The parallelization strategy of these tools is the same as the one described in Sect. 2.1.

The splitting tool (named *split*) allows to separate the original file into several ones, one per chromosome. Each of them is a valid VCF file. The statistics tool (*stats*) takes the data generated via the VCF library and performs some additional operations in order to enrich the data. As a result, the summary about a file contains the ratio between transitions and transversions or the average quality of the records in the file, among others. Finally, the filtering tool (*filter*) allows to chain several filters in order to get a set of variants that fulfill multiple conditions. Some filters take as input the results from the statistics tool. After filtering, two resulting VCF files are obtained, with the variants that passed and failed all filters, respectively.

4 VCF Handling Library

In order to set the foundations for HPG VCF Tools and HPG Variant, an efficient library for VCF file handling was implemented. Other existing applications tend to load whole files in main memory, making impossible to manage large files on systems without tens of gigabytes of RAM available per user.

VCF Library (*vcf-lib*) includes functionality for reading and writing VCF files, as well as the foundations for other tasks such as filtering and getting statistics. The file parsing has been implemented using the Ragel State Machine Compiler [13]. Similar parsers for GFF and PED files have been implemented using the same software.

Filters and statistics have been implemented as independent functions that can be easily embedded in data-parallel structures such as OpenMP for-loops. Available filters allow to remove variants not in certain regions, being (or not) SNP, having a quality or coverage lesser than specified or a number of alleles distinct than specified.

Statistics get information about each variant and a whole VCF file. The statistics of a variant show the times and relative frequency of appearance of the alleles and the genotypes than can be obtained when alleles are combined. The file statistics include the number of variants, samples, SNP, indels, bi-allelic and multi-allelic mutations, transitions and transversions in the file, and the number of variants that passed the filters applied.

5 Case Study

In this section we will illustrate a real case study using a dataset consisting of 49 trios, unaffected parents and an affected child, of sporadic short-segment Hirschsprung's diseased individuals. In total, 147 samples and 499,264 variants have been analyzed.

The analysis of sequencing data generally involves hundreds of thousands or even millions of variants. In order to avoid spurious associations in high error rate data, some quality control considerations must be noted before running the

analysis [14]. HPG VCF Tools *stats* describes variants in detail, making easier to detect low quality data. In this dataset we identified 1,092 missing variants in more than 20% of the samples and 36,428 variants with minor allele frequency (MAF) <0.5%. These variants were discarded using HPG VCF Tools *filter*.

Since the analyzed individuals have been sampled from families, the most convenient of the analysis included in HPG Variant *gwas* is a family-based association study. The transmission disequilibrium test [15] showed a significant (p -value <0.05) association of Hirschsprung's disease with 23,012 variants.

Significant variants that result from the association test are rarely the direct causal mutations. Contrarily, the association can be direct, indirect or spurious. Direct or causal associations point to variants directly involved in the disease; indirect or non-causal associations appear when the variant is neighboring the causal locus but it is not directly involved, and spurious associations occur as a result of a poor quality control or an inadequate experimental design [16].

One of the strategies to identify causal variant is filtering the associated set by those that represent a higher risk in the integrity of the genome and its products. HPG Variant *effect* identifies variants located in functionally important places of the genome and the proteins (splice sites, transcription factor binding sites, non-synonymous changes, etc.). After running the analysis we detected a high number of variants with significant association placed at the RET proto-oncogene. The vast majority of these variants were located at the coding region and produced a change in the amino acid sequence, meaning that the final protein could be significantly different in Hirschsprung's diseased individuals. Interestingly, some authors have previously confirmed the association of RET as the major gene involved in this pathogenesis [17,18], which supports the validity of our approach.

6 Experimental Results and Comparisons

In order to guarantee the possibility of running this tools efficiently even in standard workstations, performance tests have been conducted on a Dell Precision T1600 system with an Intel Xeon Sandy Bridge E3 1245 processor and 8 GB on main memory. Depending on the tool analyzed, experimental results will be shown independently or in comparison with other applications.

A great part of HPG Variant *effect* execution time is spent on I/O operations, so it would be reasonable to think that the number of samples could have a great influence on performance. In Fig. 4 it is made obvious that this is not the case for typical datasets. Fig. 4 (a) shows the same linear time complexity with 3 and 147 samples, while Fig. 4 (b) shows the mean number of variants per second processed which is, in most of the cases, in the range between 1000 and 1200 variants.

The following benchmarks were executed using 3 main threads and another 2 worker threads.

When launching TDT against a file of 500,000 variants and 147 samples it was observed that PLINK and FBAT [19] spent about 1:30 minutes, whereas HPG Variant *gwas* ran for 20 seconds. UNPHASED [20] was unable to process the dataset in a reasonable amount of time.

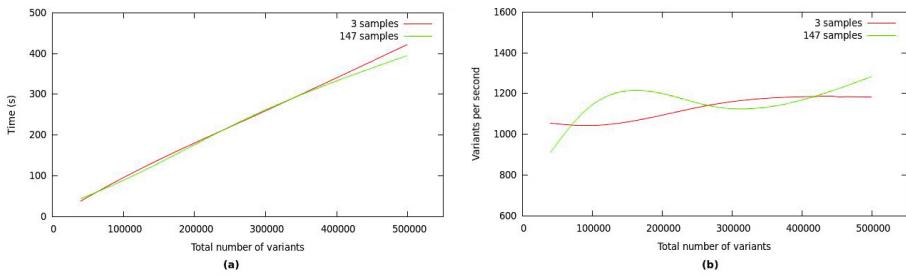


Fig. 4. HPG Variant *effect*: time spent and mean variants per second

Association test exposed similar results. PLINK and GenABEL ran for about 1:20 minutes, whereas HPG Variant spent less than 25 seconds.

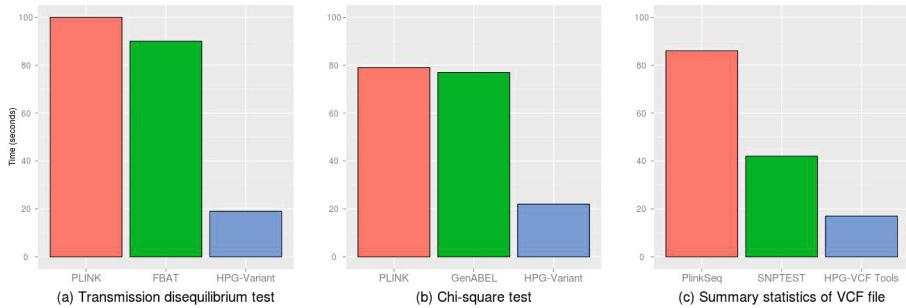


Fig. 5. Comparison of HPG Variant and HPG VCF Tools against other suites

Finally, the statistics tool from HPG VCF Tools spent about 20 seconds for the same file. PlinkSeq [21] and SNPTTEST [7] spent 90 and 40 seconds, respectively, and VCFTools could not even process the dataset after 30 minutes. Since each sample must be checked in all variants, the number of samples has a great influence over performance. When this number decreases the tool speeds-up to the extent of spending just 18 seconds processing a file of 3.5M of reads and 3 samples. For this file, VCFtools could finish its execution after 8:47 minutes.

7 Conclusions and Future Work

Next Generation Sequencing technologies produce a vast amount of data that cannot be analyzed using current software, due to its sequential nature and careless memory management. A suite that includes essential features in genomic software has been implemented, with a good performance improvement both in time and memory consumption. In addition, a freely distributable library and a set of tools for Variant Call Format handling have also been developed, so the time spent in basic and repetitive tasks could also be reduced.

As future work, more analysis such as linear or logistic regression models for variant association are being implemented. Other areas like functional and pathway analysis will also be explored.

Regarding HPG VCF Tools, the collection of filters will be enriched with more interesting, from a biological point of view, filters. The splitting tool will accept a collection of criteria instead of just the chromosome of the variant. In addition, a VCF merging utility will be developed, taking care not just of bi-allelic variants but also multi-allelic ones, a feature not well-supported at the moment in the most widespread applications.

Finally, the VCF library will allow to parse compressed VCF files, improving storage saving. The possibility of parsing indexed VCF files is also being considered.

Acknowledgements. This work is supported by the grant BIO2011-27069 from the Spanish Ministry of Science and Innovation (MICINN) and PROMETEO/2010/001 from the Consellería de Educación of the Valencian Community. We also thank the support of the National Institute of Bioinformatics (www.inab.org) and the CIBER de Enfermedades Raras (CIBERER), both initiatives of the ISCIII, MICINN. CYG is supported by the Chair in Computational Genomics funded by Bull. This work has been carried out in the context of the HPC4Genomics initiative (<http://www.hpc4g.org>).

References

1. Majewski, J., Schwartzentruber, J., Lalonde, E., Montpetit, A., Jabado, N.: What can exome sequencing do for you? *J. Med. Genet.* 48(9), 580–589 (2011)
2. Danecek, P., Auton, A., Abecasis, G., Albers, C.A., Banks, E., DePristo, M.A., Handsaker, R., Lunter, G., Marth, G., Sherry, S.T., McVean, G., Durbin, R., 1000 Genomes Project Analysis Group: The Variant Call Format and VCFtools. *Bioinformatics* 27, 2156–2158 (2011)
3. Tarraga, J., Gonzalez, C.Y., Requena, V., Dopazo, J., Medina, I.: High Performance Genomics (HPG) Project, <http://docs.bioinfo.cipf.es/projects/hpg-project>
4. Purcell, S., Neale, B., Todd-Brown, K., Thomas, L., Ferreira, M.A., Bender, D., Maller, J., Sklar, P., de Bakker, P.I., Daly, M.J., Sham, P.C.: PLINK: a toolset for whole-genome association and population-based linkage analysis. *Am. J. Hum. Genet.* 81(3), 559–575 (2007)
5. Purcell, S.: PLINK v1.07, <http://pngu.mgh.harvard.edu/purcell/plink/>
6. Aulchenko, Y.S., Ripke, S., Isaacs, A., van Duijn, C.M.: GenABEL: an R library for genome-wide association analysis. *Bioinformatics* 23(10), 1294–1296 (2007)
7. Marchini, J., Howie, B.: Genotype imputation for genome-wide association studies. *Nat. Rev. Genet.* 11(7), 499–511 (2010)
8. Medina, I., De María, A., Bleda, M., Salavert, F., Alonso, R., Gonzalez, C.Y., Dopazo, J.: VARIANT: Command Line, Web service, and Web interface for fast and accurate functional characterization of variants found by Next Generation Sequencing. *Nucleic Acids Res., Web Server Issue* 40(W1), W54–W58 (2012), <http://docs.bioinfo.cipf.es/projects/variant/wiki/>

9. Bleda, M., Tarraga, J., De Maria, A., Salavert, F., Garcia-Alonso, L., Celma, M., Martin, A., Dopazo, J., Medina, I.: CellBase, a comprehensive collection of RESTful web services for retrieving relevant biological information from heterogeneous sources. *Nucleic Acids Res., Web Server Issue* 40(W1), W609–W614 (2012)
10. Hindorff, L.A., Sethupathy, P., Junkins, H.A., Ramos, E.M., Mehta, J.P., Collins, F.S., Manolio, T.A.: Potential etiologic and functional implications of genome-wide association loci for human diseases and traits. *Proc. Natl. Acad. Sci. USA* (2009)
11. Bleda, M., Tarraga, J., De Maria, A., Salavert, F., Garcia-Alonso, L., Celma, M., Martin, A., Dopazo, J., Medina, I.: CellBase v1,
<http://docs.bioinfo.cipf.es/projects/cellbase/wiki>
12. Medina, I., De Maria, A., Alonso, R., Salavert, F., Dopazo, J.: Genome Maps, a new generation of genome browser based on HTML5. Unpublished work,
<http://genomemaps.org/>
13. Thurston, A.D.: Parsing Computer Languages with an Automaton Compiled from a Single Regular Expression. In: Ibarra, O.H., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 285–286. Springer, Heidelberg (2006)
14. Anderson, C.A., Pettersson, F.H., Clarke, G.M., Cardon, L.R., Morris, A.P., Zondervan, K.T.: Data quality control in genetic case-control association studies. *Nat. Protoc.* 5, 1564–1573 (2010)
15. Ott, J., Kamatani, Y., Lathrop, M.: Family-based designs for genome-wide association studies. *Nat. Rev. Genet.* 12, 465–474 (2011)
16. Clarke, G.M., Anderson, C.A., Pettersson, F.H., Cardon, L.R., Morris, A.P., Zondervan, K.T.: Basic statistical analysis in genetic case-control studies. *Nat. Protoc.* 6(2), 121–133 (2011)
17. Bolk Gabriel, S., Salomon, R., Pelet, A., Angrist, M., Amiel, J., Fornage, M., Attie-Bitach, T., Olson, J.M., Hofstra, R., Buys, C., Steffann, J., Munnich, A., Lyonnet, S., Chakravarti, A.: Segregation at three loci explains familial and population risk in Hirschsprung disease. *Nature Genet.* 31, 89–93 (2002)
18. Emison, E.S., McCallion, A.S., Kashuk, C.S., Bush, R.T., Grice, E., Lin, S., Portnoy, M.E., Cutler, D.J., Green, E.D., Chakravarti, A.: A common sex-dependent mutation in a RET enhancer underlies Hirschsprung disease risk. *Nature* 434, 857–863 (2005)
19. Laird, N., Horvath, S., Xu, X.: Implementing a unified approach to family based tests of association. *Genet. Epidemiol.* 19(suppl. 1), S36–S42 (2000)
20. Dudbridge, F.: Likelihood-based association analysis for nuclear families and unrelated subjects with missing genotype data. *Hum. Hered.* 66, 87–98 (2008)
21. Purcell, S.: PLINK/SEQ v0.08, <http://atgu.mgh.harvard.edu/plinkseq/>

A Novel Implementation of Double Precision and Real Valued ICA Algorithm for Bioinformatics Applications on GPUs

Amin Foshati¹ and Farshad Khunjush^{2,3,4}

¹ Department of Computer Science and Engineering,
International Division of Shiraz University, Shiraz, Iran
a.foshati@gmail.com

² Department of Electrical Engineering, Hormozgan University, Bandar-Abbas, Iran
³ School of Electrical & Computer Engineering,
Department of Computer Science and Engineering, Shiraz University, Shiraz, Iran
⁴ School of Computer Science,
Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
khunjush@cse.shirazu.ac.ir

Abstract. Several applications in the field of bioinformatics require extracting individual source signals from a large amount of observed data (signal mixtures). Among the available solutions, a possible approach is the independent component analysis (ICA). However, this computationally intensive algorithm does not fit for many real-time or large size data applications. As a result, this shortcoming calls for speeding up the execution of this algorithm. Recently, graphics processing units (GPUs) have emerged as general-purpose parallel processing accelerators. This platform has the potentials to be leveraged in processing a large amount of signals received from medical devices such as EEG and ECG tools. This work provides the implementation of an ICA algorithm, Joint Approximate Diagonalization of Eigen-matrices (JADE), on a low cost programmable graphics cards using CUDA programming toolkits. For this implementation, we achieved an overall speedup of over 7.9x for estimating 64 components, each with 9760 samples.

Keywords: Independent Component Analysis (ICA), JADE Algorithm, Blind Source Separation (BSS), Graphics Processing Units (GPUs), CUDA, Bioinformatics Applications.

1 Introduction

There are many bioinformatics applications and clinical medical tools such as EEG and ECG tools, which record the brain and heart electrical activities, need to process a large amount of signals that they receive. The amount of data that needs to be processed in some cases is too much that requires a high-performance computing environment to be able to handle the data accurately and quickly. Signal processing techniques play key roles in these types of applications and need especial attentions.

Independent Component Analysis (ICA) is one of the most widely used algorithms for signal processing applications particularly in bioinformatics domains. The practical and popular use of ICA is the Blind Source Separation (BSS). This method is applied to recover individual signals from a combination of signals where there is no prior knowledge. There have been a number of studies showing the applications of ICA in a variety of domains such as medicine, economics, and engineering.

There exist various ICA methods implementations such as FastICA [1], InfoMax [2], and JADE [3] for the BSS problem. Among these, we have considered the JADE algorithm to investigate its performance when GPUs are used as the computation engine. One reason for this selection has been its potentials in being run in parallel as a result of using Jacobi rotation [4] for joint diagonalization. Another reason for this selection is that this algorithm supports Gaussian signals, which are not applicable in the FastICA method; moreover, parameters in the JADE algorithm do not need to be tuned. As a result, we have decided to implement a parallel version of this method that can be heavily used in bioinformatics and medicine.

Recently, Graphics Processing Units (GPUs) have demonstrated staggering computational power in applications that can leverage parallel environments. These low cost accelerators open up new opportunities for speeding up signal and image processing applications and algorithms instead of using traditional complex Digital Signal Processors and FPGAs. “The programmable GPU has evolved into a highly parallel, multithreaded, and many core processors with tremendous computational horsepower and high memory bandwidth” [5]. We have chosen the GPU architecture for implementing our parallel version of the JADE algorithm.

CUDA™, which is a general purpose parallel computing architecture, has been introduced by NVIDIA to leverage the parallel compute engines in NVIDIA GPUs for solving computationally intensive problems faster than a CPU [5]. CUDA extends the C language by adding some features that satisfy this parallel programming model. CUDA has been employed for the parallel implementation of the JADE algorithm on GPU.

As stated, the goal of this work is to provide a novel implementation for a double-precision and real-valued ICA algorithm on GPUs. There have been several implementations of the ICA algorithm on GPUs [6-12]. Most of these studies have proposed single precision solutions. In this work, we focus on the JADE algorithm. To the best of our knowledge, the only single precision implementation of this algorithm has been proposed in [7], which has 2 restrictions: first, the maximum number of sweeps was set to 100; second, the maximum number of components (channels) is limited to 50. Our experiments show in some cases the JADE algorithm is quite sensitive to the precision of operations. This problem is originated from eigenvalues of the covariance matrix. If the value of an eigenvalue approaches zero, the result changes significantly. For example, in our implementation a small change in the eigenvalues changes the result from 65 to 132. This phenomenon calls for a higher precision implementation for applications requiring more accuracy.

The rest of this paper is organized as follows. Section 2 presents the background of the ICA method and the JADE algorithm. The GPU architecture is also discussed briefly in this section. Section 3 discusses the approach for implementing different

components in the JADE algorithm. Our methodology is elaborated in section 4. The results are presented in section 5. Finally, section 6 concludes the paper and discusses our future work.

2 Background

2.1 Basics

In this part, we explain the ICA method, which attempts to model a set of signals as a linear combination of source signals that are hidden among observable signals. The original signals (i.e., source signals) are assumed to be statically independent from each other. For example, x_1, x_2, \dots, x_n and s_1, s_2, \dots, s_n signals are observed and source signals and the coefficients of $a_{11}, a_{12}, \dots, a_{ij}, \dots, a_{nn}$ are the weights of the j_{th} observed signals collected by i_{th} sensors. In this situation, the mixed source signal equations can be expressed as follows:

$$\begin{aligned}x_1(t) &= a_{11}S_1(t) + a_{12}S_2(t) + \dots + a_{1n}S_n(t) \\x_2(t) &= a_{21}S_1(t) + a_{22}S_2(t) + \dots + a_{2n}S_n(t) \\&\vdots \\x_n(t) &= a_{n1}S_1(t) + a_{n2}S_2(t) + \dots + a_{nn}S_n(t)\end{aligned}\quad (1)$$

We can transform the equation set 1 to a matrix form, that is, $X = AS$. In this equation, the X vector (observed signals) is known, but both S and A are unknown. The ICA method attempts to find the inverse of A matrix named W , and multiply W by X to find the estimated source signals called Y .

For example, Fig. 1 presents 3-channel signals mixed with a 4×3 matrix and an added small Gaussian noise. The figure also shows the separation of signals using the JADE algorithm. This example was resulted from running Cardoso's MATLAB demo of JADE real valued algorithm (This code is available on Cardoso's webpage).

2.2 The JADE Algorithm

As stated, the JADE algorithm is chosen for parallel implementation as it has several independent components that can be run in parallel. Cardoso and Souloumiac proposed this method in 1993 [3]. The JADE algorithm strictly depends on the memory capacity of the system on which it is running; thus, component numbers are limited by the computer memory. This algorithm includes the following steps:

1. Removing the mean value of each component in the observed matrix (X), calculating covariance matrix, computing whitening matrix from covariance matrix, and rescaling the X (preprocessing step).
2. Calculating the maximal set of Cumulant Matrix (CM).
3. Applying Jacobi rotations in CM for joint diagonalization.
4. Estimating the final result.

In step 1, the observed data (\mathbf{X}) is converted to zero-mean and white matrix. For whitening, eigenvalue and eigenvector decomposition are used. In step 2, this algorithm is used forth-order cumulant matrix. These matrices are symmetric and the number of them are $n*(n+1) / 2$. In step 3, the cumulant matrices are diagonalized by applying Jacobi rotations that described in section 3.2. The rotation matrix is used to transform the results for estimating the inverse of mixing matrix (\mathbf{A}) which is called \mathbf{W} (Step 4).

In the next sections, we explain the details of steps that have been implemented on the GPU. The following section briefly describes the GPU architecture.

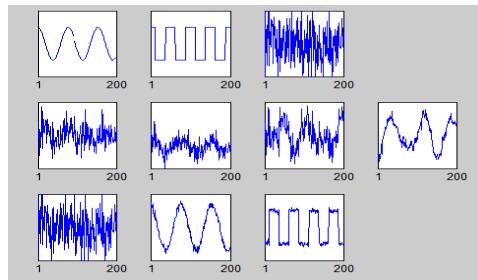


Fig. 1. The example of running the JADE algorithm (First row: three source signals: a sine wave, a square wave, a white Gaussian process. Second row: the result of mixing by a 4×3 matrix and adding a small Gaussian noise. Third row: the result of JADE processing).

2.3 GPU

In recent years, Graphics Processing Units (GPUs) have evolved from special-function devices to massively parallel and general-purpose programmable processors. GPU performance has been increasing at a rate of 2.5x to 3.0x annually [13]. NVIDIA GeForce GTX570 is one example of GPUs employed in this study. It includes three billion transistors supporting 480 1.46 GHz processing units, which are organized in 15 streaming multiprocessors. The GPU's theoretical peak performance is reported over 1.4 Teraflops. NVIDIA GPUs are programmed using CUDA, which is expressed as an extension to C and C++ languages.

3 Implementation

To implement the JADE algorithm, in the first step our main concern was to determine the sections of the algorithm that take more time. For this, we profile the best serial implementation of the JADE algorithm and the results are shown Table 1. We employed the serial code of the JADE algorithm provided by Cardoso and used the configuration as stated in the Experimental results section. Then, the optimized version of this algorithm is considered as the baseline. For this purpose, the DGEMM function is employed from the BLAS [14] library for multiplication and the DSYEV function in LAPACK [15] library for computing eigenvalues and eigenvectors.

As indicated, this algorithm includes four steps. Our profiling reveals that steps 1 and 4 have low execution times in comparison to others; therefore, we focus on steps 2 and 3 in this paper.

Table 1. Profiling Results for 64 channels and 9760 samples from “S001R01.edf” file [20, 21]

Steps	1	2	3	4
Time(milliseconds)	52.58	92275.21	1127163.35	33.08
Percentage of time	0.0043%	7.567%	92.426%	0.00271%

3.1 Cumulant Matrices

Cumulants of data are non-linear combinations of different moments. For example, if we consider X as a real-valued, zero-mean, continuous scalar random variable, the first characteristic function of X will be defined as the continuous Fourier transform of the probability distribution function [16]. In addition, every probability distribution can be specified by its characteristic function. Moments of data are the coefficients of the Taylor series expansion of the first characteristic function and cumulants are the coefficients of the expansion of the second characteristic function. The cumulant forth-order matrix equation used in the JADE algorithm is shown in Eq. 2.

$$\begin{aligned} \text{Cum}(X_i, X_j, X_k, X_l) = & E(x_i x_j x_k x_l) \\ & - E(x_i x_j) E(x_k x_l) \\ & - E(x_i x_k) E(x_j x_l) \\ & - E(x_i x_l) E(x_j x_k) \end{aligned} \quad (2)$$

The X variable is whitened so its variance is equal to one, if X_i is equal to X_j , $E(x_i x_j) = 1$ and otherwise $E(x_i x_j) = 0$. Therefore Eq.2 can be divided into:

$$\text{Cum}(X_i, X_j, X_k, X_l) = \begin{cases} E(x_i x_j x_k x_l) - 3 & i = j = k = l \\ E(x_i x_j x_k x_l) - 1 & i = j \text{ and } k = l \\ E(x_i x_j x_k x_l) - 1 & i = l \text{ and } k = j \\ E(x_i x_j x_k x_l) - 1 & i = k \text{ and } j = l \\ E(x_i x_j x_k x_l) & \text{otherwise} \end{cases} \quad (3)$$

If the observation matrix contains n random variables (channels) with T observations, the JADE algorithm will require $(n * (n + 1)) / 2$ cumulant matrices and the size of each matrix will be equal to $n * n$. This implies that the memory requirement for JADE is $O(n^4)$; therefore, the scalability of the algorithm depends on the memory of the running system.

As the computation of each cumulant matrix is independent from each other this part can be parallelized. We employ a customized kernel to initialize the computation of expected values. The DGEMM function from CUBLAS [17] level-1 library is tuned for matrix multiplication to find expected values. At the end of the computations, a customized kernel computes the final result. Because observed signals (X) and cumulant matrices require a large amount of memory, we are not able to use

on-chip shared memory of streaming processors, which is very limited (e.g., for n equal to 76 and T equal to 10000, the required memory is about 140 megabyte).

3.2 Joint Diagonalization

The JADE algorithm uses Jacobi idea to join the diagonalization cumulant matrices. This method includes an iterative approach of applying Given's rotations to cumulant matrices to converge the algorithm. Cardoso et al. propose the generalized cyclic version of the Jacobi algorithm for this method, as shown in Fig.2. When all the values of theta within a sweep are lower than a threshold, set by default to the square root of the machine precision, the execution stops. The $G(p, q, c, s)$ is a Given's rotation matrix form as shown in Fig.3. The $G(p, q, c, s)^*CM$ and $R^*G(p, q, c, s)$ are named as left rotation, and $CM^*G^T(p, q, c, s)$ is known as right rotation.

It should be noted that each rotation needs 3 matrix multiplications, but we decided not to use neither DGEMM function, because of the sparsity of $G(p,q,c,s)$, nor CUSPARSE [18] library. We use our kernel for rotating cumulant matrices as a better implementation.

In this implementation, we also require synchronization for already existing dependencies between right and left rotations of the cumulant matrix. To achieve this, we assign each cumulant matrix to a block and assume threads for each block within the number of random variables (channels). Using the shared memory is not possible due to the iterative feature of this algorithm and using the results of the cumulant matrix in the next kernel call. As shown in Fig. 2, the calculated value of theta in the GPU must be transferred to the host for testing its convergence. In additions, asynchronous communication is employed to hide the latency of memory transfers.

```

Input : Cumulant Matrices (CM)
Output : Unitary Diagonalizer (R)

R ← In
Tr ← Compute Threshold

While (Θ > Tr) do
    For p ← 1 to n - 1 do
        For q ← p + 1 to n do
            Compute Θ
            c, s ← Compute Sin(Θ) and Cos(Θ)
            CMi ← G(p, q, c, s) CMi GT(p, q, c, s)
            R ← RG(p, q, c, s)

```

Fig. 2. A pseudo code of joint diagonalization algorithm

$$\begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix}$$

Fig. 3. The definition of $G(p,q,c,s)$ matrix ($p < q$)

The joint diagonalization algorithm calls two kernels in every iteration, rotation and compute Θ . The theta computation needs last changes of the cumulant matrices to calculate theta and to apply a reduce technique [19] to sum up the results.

In the following section, we describe our methodology and assumptions.

4 Methodology

In these experiments, we have employed a system consisting of a Core-i5 processor running at 3.1 GHz and a NVIDIA Geforce GTX 570 GPU. This model of GPU includes 480 cores each running at 1.46GHz clock and 1.25 GB of DRAM with a maximum bandwidth of 152GB/s. We used the CUDA toolkit 4.1, MSVC-2010, and C/C++ compiler.

We also used data sets from PhysioNet [20, 21] consisting of over 1500 one and two-minute EEG signals, recorded using the BCI 2000 system [22]. We selected one minute EEG signal with 64-channel and 9760 samples from the “S001R01.edf” file. We ran all test cases 3 times and averaged the execution times. Each test case consists of two input parameters, the number of variable (channels) and samples. The sample was set to 9760 and the number of variables varies from 4 to 64 with step 4.

For testing our parallel implementation, the best serial implementation is employed as a baseline. In our experiments, the speedup is reported as the ratio of execution time of our best serial implementation to the parallel execution times.

5 Experimental Results

In this section, we present the result of our experiments. As mentioned above, for this implementation, we first optimized different components of the JADE algorithm separately. Then, we explore the effect of these components synergistically on the performance of this algorithm.

The GPU utilization and memory coalescing are two optimization techniques used for optimizing running programs on this hardware. For example, developers can break down the computations into thread blocks that are distributed throughout the GPU. This optimization is named GPU utilization. The access of memories is configured as one memory segment, which in turn reduces memory transactions. This optimization is known as memory coalescing.

5.1 Computing Cumulant Matrix Optimization

In this part, we present the achieved speedups as a result of optimizing cumulant matrices computations. This component is parallelized using the CUBLAS DGEMM function and two other customized kernels. The GPU utilization is the optimization technique that is considered in this implementation. Fig.4 depicts the relative speed up of each test case in cumulant matrices. The cumulant matrices computations are done in parallel, which is the main reason for the achieved speedups in our implementations.

The curve's fluctuation in Fig.4 was originated from increases in the number of blocks in our GPU implementations. It should be noted that if the number of blocks increases more than the occupancy limit of streaming processors, some blocks wait for the completion of other tasks to start their computations.

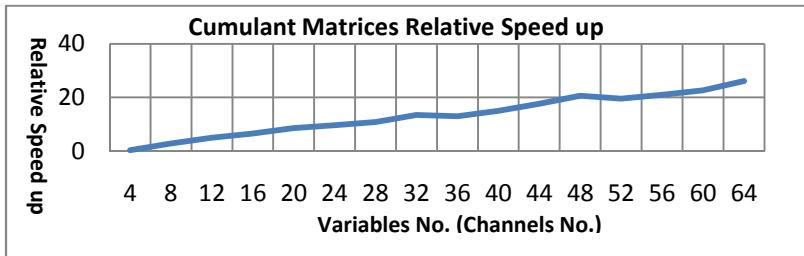


Fig. 4. The relative speed up of cumulant matrices component

5.2 Joint Diagonalization Optimization

Another component that we need to optimize is the joint diagonalization. This module uses two customized kernels to utilize the GPU and to coalesce memory accesses for its optimizations. In this part, we present the speedups as the results of the applied optimizations.

Fig.5 shows the relative speedup of each test case in the joint diagonalization module. As stated before, the curve fluctuation is the result of increasing the block numbers in the GPU implementation. Another reason is the rotation numbers (rotations are equal to the number of left or right rotations) of joint diagonalization. In other words, rotations grow more heavily when the number of variable increases. This is mainly due to having more kernel calls.

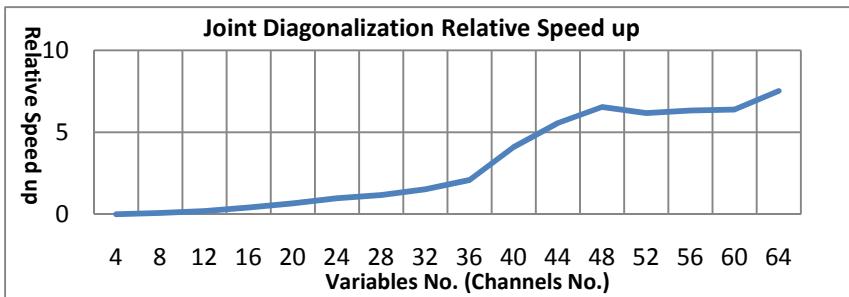


Fig. 5. The relative speed up of joint diagonalization component

5.3 Overall Performance

Having optimized the separate components, we applied the optimized versions of the above-mentioned components to the original JADE algorithm. The overall performance of our program is presented in Fig.6. As shown in this figure, in test cases with 24, 28 and 32 channel numbers (variable number), the performance degrades.

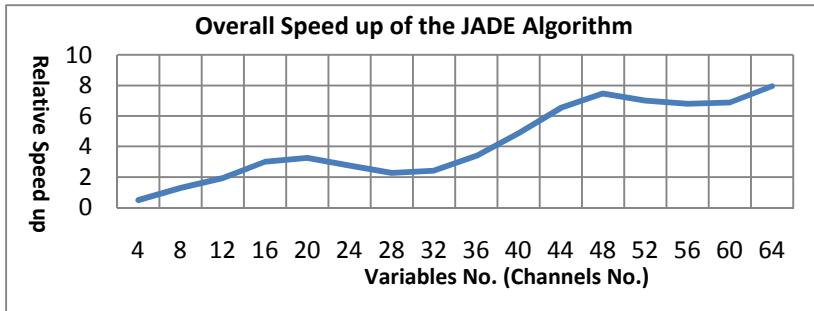


Fig. 6. The overall speedup of the JADE algorithm

This behavior is due to this fact that the joint diagonalization is a dominating component of these test cases and the overall speedup is limited by this component. As shown, we face the same scenario in channel numbers 52, 56 and 60.

It is worth mentioning that our implementation is scalable to the number of channels. As shown in Fig. 6, when the number of channels increases, the speedup increases as well. As stated before, this algorithm strictly depends on the memory capacity of the system on which it is running; thus, we have to find the number of components to compute by available memory. In GTX570, 1.25 GBs of DRAM memory is available to compute the JADE algorithm. Cumulant matrices limit the memory capacity of this algorithm. These matrices are required $n*(n+1)/2 * n*n$ words of memory that the length of each word is 8B which results in computing 128 channels by this GPU.

6 Conclusion and Future Work

This paper presented the implementation of a double precision ICA on GPUs. As part of this work, we developed the first double precision parallel cumulant matrices and joint diagonalization based on Jacobi rotation modules. The proposed JADE implementation achieved the maximum overall speedup of over 7.9x for estimating 64 sources, each with 9760 samples. On the other hand, the maximum speedups of joint diagonalization and cumulant matrices are 26.14 and 7.52, respectively. As the future work, we plan to decrease the number of CUDA kernel calls and at the same time try to use the CPU to improve the whole performance.

Acknowledgments. The authors thank Dr. Reza Sameni and Dr. Reza Azimi for their helpful comments and suggestions. This work was supported in part by the IPM under grant number 90/80/5813.

References

1. Hyvärinen, A., Oja, E.: Independent Component Analysis, Algorithms and Applications. *Neural Networks* 13(4-5), 411–430 (2000)
2. Bell, A.J., Sejnowski, T.J.: An Information-maximization Approach to Blind Separation and Blind Deconvolution. *Neural Computation* 7(6), 1129–1159 (1995)

3. Cardoso, J., Souloumiac, A.: Blind Beamforming for Non-Gaussian Signals. IEE Proceedings. Part F. Radar and Signal Processing 140(6), 362–370 (1993)
4. Févotte, C., Theis, F.J.: Orthonormal Approximate Joint Block-diagonalization. Technical Report (2007)
5. NVIDIA Corporation: NVIDIA CUDA C Programming Guide. Version 4.1 (2011)
6. Forgette, J., Wachowiak-Smolíková, R., Wachowiak, M.: Implementing Independent Component Analysis in General-Purpose GPU Architectures. In: Snasel, V., Platos, J., El-Qawasmeh, E. (eds.) ICDIPC 2011, Part II. CCIS, vol. 189, pp. 233–243. Springer, Heidelberg (2011)
7. Brandt, D.: Investigation of GPGPU for Use in Processing of EEG in Real-time. Master Thesis (2010)
8. Ramalho, R., Tomás, P., Sousa, L.: Efficient Independent Component Analysis on a GPU. In: 2010 IEEE 10th International Conference on Computer and Information Technology, CIT, pp. 1128–1133 (2010)
9. Forgette, J., Wachowiak-Smolíková, R., Wachowiak, M.: Scalable Parallel Implementation of Independent Components Analysis on the Graphics Processing Unit. In: 2011 24th Canadian Conference on Electrical and Computer Engineering, CCECE, pp. 912–916 (2011)
10. Liang, T.-Y., Wang, T.-H.: A Cloud Computing Service for Fast Audio Source Signal Separation. In: 2011 IEEE International Workshop on Machine Learning for Signal Processing, MLSP, pp. 1–6 (2011)
11. Raimondo, F., Kamienkowski, E.J., Sigman, M., Slezak, D.F.: CUDAICA: GPU Optimization of Infomax-ICA EEG Analysis. Computational Intelligence and Neuroscience 2012, Article ID 206972 (2012)
12. Jiang, C., Li, P., Luo, Q.: High Speed Parallel Processing of Biomedical Optics Data with PC Graphic Hardware. In: Proc. of SPIE-OSA-IEEE Asia Communications and Photonics, SPIE, vol. 7634, p. 76340U (2009)
13. Himawan, B., Vachharajani, M.: Deconstructing Hardware Usage for General Purpose Computation on GPUs. In: Fifth Annual Workshop on Duplicating, Deconstructing, and Debunking (in Conjunction with ISCA-33) (2006)
14. Blackford, L.S., Demmel, J., Dongarra, J.J., Duff, I.S., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K.: An Updated Set of Basic Linear Algebra Subprograms (BLAS). ACM Trans. Math. Soft. 28(2), 135–151 (2002)
15. Anderson, E., Bia, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK users' guide, 3rd edn. SIAM (1999)
16. Hyvärinen, A., Karhunen, J., Oja, E.: Independent Component Analysis. John Wiley & Sons (June 1, 2001)
17. NVIDIA Corporation: CUBLAS Library. CUDA Toolkit 4.1 (January 2012)
18. NVIDIA Corporation: CUSPARSE Library. CUDA Toolkit 4.1 (January 2012)
19. Harris, M.: Optimizing Parallel Reduction in CUDA. CUDA Webinar, NVIDIA Corporation (2007)
20. Goldberger, A.L., Amaral, L.A.N., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.: PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. Circulation 101(23), e215–e220 (2000)
21. EEG Motor Movement/Imagery Dataset,
<http://www.physionet.org/pn4/eegmmidb/>
22. Schalk, G., McFarland, D.J., Hinterberger, T., Birbaumer, N., Wolpaw, J.R.: BCI2000: A General-Purpose Brain-Computer Interface (BCI) System. IEEE Transactions on Biomedical Engineering 51(6), 1034–1043 (2004)

PROGENIA: An Approach for Grid Interoperability at Workflow Level

Maria Mirto¹, Marco Passante², and Giovanni Aloisio^{1,2}

¹ Euro-Mediterranean Center on Climate Change (CMCC), Italy

² University of Salento, Lecce, Italy

Abstract. This paper addresses the problem of simulating complex large-scale experiments by using the PROGENIA WorkFlow Management System (WFMS), developed in the ProGenGrid (Proteomics and Genomics Grid) research project at the University of Salento, deployed and tested in a real Grid-based Problem Solving Environment for Bioinformatics named LIBI (International Laboratory of Bioinformatics). PROGENIA aims to achieve interoperability at workflow level, supporting the deployment of a workflow on different Grids based on Globus, gLite and Unicore middlewares. By using specific adapters, the workflow engine acts as a meta scheduler, submitting the jobs on different grids. The meta scheduler selects the available resources from a list of resources, previously configured by the PROGENIA administrator, by using the interfaces of the PROGENIA editor to configure a grid (Users admin, Virtual Organization, Resource and Software Management). PROGENIA will allow domain researchers to share and reuse their scientific workflows across distributed computing infrastructure. PROGENIA has been tested in several Bioinformatics case studies. In particular a test case related to the protein multi-alignment, executed on gLite and Globus middleware will be presented.

1 Introduction

The resources needed to execute workflows in a Grid environment are commonly highly distributed, heterogeneous, and managed by different organizations. One of the main challenges in the development of Grid infrastructure services is the effective management of those resources, in such a way that much of the heterogeneity is hidden from the end-user. This requires the ability to orchestrate the use of various resources of different types. In recent years, various distributed computing infrastructures (DCIs) have been developed to support national and international research activities. These “grid production” infrastructures such as EGEE (now EGI)[5], DEISA [4] and TeraGrid allow the sharing of heterogeneous resources in order to support large-scale experiments. However, in spite of efforts made by the grid community, the interaction between these grids is still limited. The main reasons are the diversity of the adopted middleware, diversity in data management, authentication and authorization, security policies and middleware versions. This makes it very difficult for domain researchers developing their application for a certain DCI to benefit from other implementations available in other DCIs.

In this sense, the Open Grid Forum (OGF) [13] has developed open standards for grid software interoperability, while the OGFs Grid Interoperation Now - Community Group (GIN-CG) [7] is coordinating a set of interoperation efforts among production grids, since interoperability requires definition, introduction and implementation of standards.

DCIs are fundamental for managing the deluge of biological data produced by large-scale experiments such as genome projects. Biological datasets present challenges from many different perspectives. The scale of data to be processed requires access to large collections of widely distributed resources such as those found in computational grids.

The analysis of this data needs to be farmed out to hugely provisioned computing resources with large and efficient storage devices. In fact, for their daily analyses of genomes, bioinformaticians need efficient access to these biological data and bioinformatics programs. Moreover, these datasets are not static: as discoveries are made, new entries are added to the database and existing ones are updated. Complicating the problem of data management is the fact that most bioinformatics applications are not designed with grid computing in mind. In fact, many valuable applications were designed, tested, and validated long before grid computing arose. As a result, such applications are designed to perform simple local I/O and have no facility for attaching to grid data systems. Nevertheless, these applications need both high-throughput computing and huge data storage [3][8]. Workflow technology is important to facilitate user-friendly access to huge amount of resources and offers a standard model to formalize an experiment. Moreover, workflow interoperability increments exchange of existing workflows between users, workflow systems and DCIs.

A WorkFlow Management System (WFMS), named PROGENIA, has been implemented at the University of Salento which aims at providing a software platform where e-scientists can simulate experiments composed of existing analysis and visualization tools, monitoring their execution, storing the intermediate and final output and finally, if needed, saving the model of the experiment for updating or reproducing it. The tools that we have considered are software components and data resources wrapped as services and composed through a workflow [10], [11]. PROGENIA has an editor for composing and monitoring the workflow of jobs and an engine for scheduling the jobs in a Computational Grid through Globus [6], gLite [9] and Unicore [1] middleware. In this paper we present a test case related to the protein multialignment problem, executed on gLite and Globus middleware through PROGENIA. The rest of the paper is organized as follows. Section 2 describes used approach for interoperability at a workflow level hence Section 3 discusses a case study on protein multi alignment sequences. Finally Section 4 describes conclusions and future work.

2 Workflow Interoperability Approach

To allow the interoperability between different grid middleware, a Grid Meta Scheduler (GMS) has been developed. Such service that interfaces with multiple

local schedulers which control the use of individual nodes, by negotiating with them advance reservation of resources, based on user requirements such as time or QoS constraints. The goal of this negotiation is to determine feasible time slots in which all required resources are available for the requested start time to execute the distributed workflow. The main functions of a meta scheduler include (i) allocation of a single resource for a single application for a fixed period of time, (ii) co-allocation of multiple resources for the same fixed period of time for single or multiple applications, (iii) allocation of multiple resources for multiple applications for different fixed periods of time, and (iv) allocation of dedicated resources for either of the cases above.

The GMS has been designed taking into account the following requirements:

- ability to handle simple workflows described by DAGs;
- ability to handle complex workflows, described by arbitrary graphs, supporting cycles and conditions;
- support for recursive composition, i.e. the possibility to define a workflow vertex as a sub-workflow or parameter sweep vertex instead of a batch task;
- support for secure remote execution on different grid middleware and not:

2.1 The GMS Meta Scheduler

The GMS is a multi threaded meta scheduler. It presents a web service interface developed using gSOAP with GSI-enabled plug-in for gSOAP. It uses also middleware libraries (adapters) to access Grid resources and services provided by the underlying grid middlewares.

An important aspect is the possibility to track the history of the job in each of its status: whereas Globus returns all status of a job, through the Globus JobManager, gLite and Unicore do not have this feature, returning just the current status. Our approach consists of a polling process of the GMS that is run periodically in order to update/add the status into an internal database that manages the queuing of jobs.

One of main features of PROGENIA is possibility to handle complex workflows described by arbitrary graphs, supporting cycles and conditions and recursive composition: a complex mechanism was implemented to handle workflow jobs. The management of these jobs is made considering the vertices that are on the highest level of the graph not yet explored and for each vertex found, GMS initializes its counter variable (*visits*) to zero and then determines the number of its active inputs, that represents a single file or a directory needed as input by a vertex.

The GMS produces a set of ready tasks that can safely be queued for execution; a ready task is immediately handled by spawning a new thread in charge of task execution. In particular, this thread queues its task in an internal database.

Job submission is synchronous. Each thread waits until there are children tasks that must be scheduled in the workflow. The GMS allows job rescheduling up to

a maximum of three times after which all workflow will definitely be failed and its status set to be cancelled.

Upon job completion, the GMS thread increments the *visits* variable. If this vertex is an active input for one or more condition vertices then the associated conditions are also evaluated.

This element allows adding a control flow based on the truth of expressions related to conditions specified by the user. A condition vertex has a unique input connector, but can have many output connectors; it is characterized by (i) a condition type, (ii) evaluator, (iii) result type and optionally (iv) a file based condition input connector.

The type of a condition can be internal or file based. Internal conditions refer to the internal state of a vertex, in particular the number of times the vertex has been visited so far.

File based conditions allow evaluating the file content related to a specific input connector. The result type indicates how the result of the condition must be interpreted; in particular this attribute can be used to specify numeric or string types. The file based condition input connector denotes an input connector and must be specified only in case of a file based condition. For each output connector there is a corresponding condition to be evaluated. Many conditions can be satisfied simultaneously; for each condition actually satisfied, the input connector is replicated in the corresponding output connector. Condition vertices easily allow defining a switch construct. The files produced as output are transferred to next execution vertex as input files. For each output connector, the handling thread modifies the corresponding input connector and decrements the number of active inputs of the next execution or condition vertex. If, after examining all of the output connectors, the next vertex has no active inputs, a new thread is spawned to handle the execution of the next vertex. The main thread responsible for all workflow submissions, waits until there are no remaining tasks to be scheduled in the workflow.

Parameter sweep jobs within the workflow are automatically handled without explicit user intervention. When a subtask produces as output a directory and the successive subtask takes as input a file, whose name is a parameter indicated as \${filename} that has not been specified in the JSDL description, the GMS automatically instantiates the successive subtask as many times as the number of files in the output directory of its preceding subtask and it proceeds to queuing step.

If the job extracted from queue is batch, a scheduling process starts. The GMS determines between all resources specified in the JSDL file belonging to target middleware those not currently in use in the system or those not used during other resubmissions of the current job, then it acquires the delegated credential(s) from the specified MyProxy server and submits job utilizing the corresponding Job Submission Adapter.

All of the GMS actions are logged to stable storage for subsequent accounting and auditing.

2.2 Job Submission Adapters

The modular architecture of the GMS allows possible extensions through a plug-in based approach (see Figure 1). Owing to the plugged adapters, the GMS can contact different remote resource managers providing the support for multiple and heterogeneous Grid services. In particular, GMS contains several adapters which using libraries that we have developed, provide core functions for job submission/monitoring and data transfer by using the GridFTP protocol.

The Globus adapter takes as input JSDL files and translates them into RSL language. It submits batch jobs on a Globus remote machine, providing support for automatic staging of executable and input file(s). Moreover, the output file(s) can also be transferred automatically upon job completion to target machine. It also includes a control service for checking the status of submitted jobs, contacting remote Globus JobManager to inquire the information about it. Querying GRAM returns the complete history state of a job. Because computational grids based on Globus Toolkit requires users authentication, the adapter implements a restricted form of delegation, with X.509 users certificate, a proxy which acts on behalf of the user, valid for a specified number of hours. For batch submission, the library uses the GRAM APIs. The submission function is non blocking, i.e. it submits the job and returns immediately. It provides, also, additional support for automatic staging of executable and input/output file(s) using GridFTP protocol.

gLite adapter is composed by two libraries implementing the web service clients of the Workload Management System (WMS) and CREAM for the submission of a batch or workflow job. The GMS can access the WMS using the user's credential with the VOMS extension got from the VOMS server. The job is described according the JDL language and submitted to the WMPProxy web service using GSI protocol. The job monitoring is performed contacting the Logging and Bookkeeping Service (L&B) web service whereas the file stage-in and stage-out is explicitly done via GridFTP contacting the appropriate Storage Element (SE). The CREAM library supports typical job management capabilities of submission, cancellation and the ability to pause and resume jobs (where supported by the underlying batch system). Job state information is available through the CE Monitoring (CEMON) service which also provides an asynchronous call-back mechanism to clients.

The security is ensured by use of the GSI plug-in and for data transfer, the GridFTP protocol is used. For job definition, editor uses the standard JSDL. However, the current gLite infrastructure is highly dependent on JDL, although only at internal level. Hence, the choice to convert JSDL files into specific JDL and use gLite infrastructure with the standard JDL when required, has been exploited.

In order to develop the library for the local transformation of JSDL request in a compatible format, there was a complete semantic study about JSDL elements and then the creation of correspondences JSDL2JDL "code" regarding: (i) job identification requirements; (ii) resource requirements (matchmaking process); (iii) data requirements.

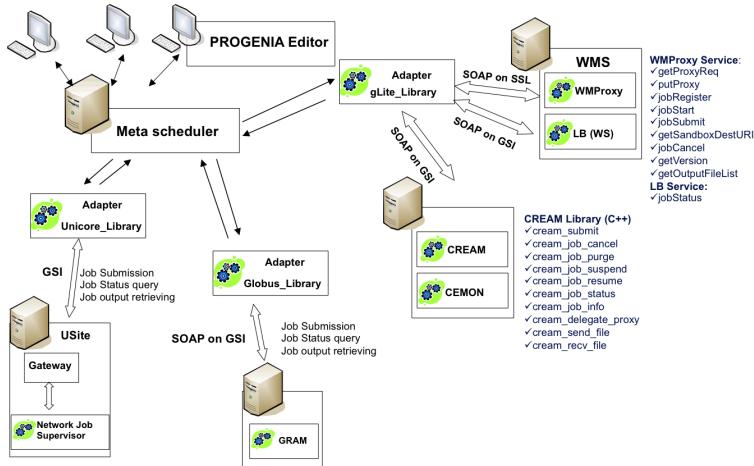


Fig. 1. GMS adapters for Grid interoperability

The gLite adapter implements a converter, based on XSLT, to translate a JSIDL file in a JDL format. Because the remote resource manager takes in input text files, the XSLT solution has been considered. Indeed, through appropriate stylesheets it is possible to transform XML files into text files (in JDL format). Being template based, XSLT is more resilient to changes in documents than low level DOM and SAX, and XSLT templates are based on XPath pattern which is very powerful in terms of performance to process the XML document.

Within the GMS, the Unicore adapter has been also developed, in order to support Unicore based grid infrastructure. The GMS can access the USite gateway service using the users credential through a GSI connection in order to submit batch jobs. The job is submitted to the Gateway using the AJO description file format. The USite Gateway exposes also functionality for job monitoring and output retrieval: the file transfer mechanism is performed through GridFTP connection with the remote USite. The Unicore adapter is a java library developed for the submission and monitoring of a job. It includes the client of the Unicore NJS (Network Job Supervisor) service. A wrapper in C, through Java Native Interface (JNI) functions, has been added and included in the GMS.

3 Case study: High throughput Protein Similarity Searches

Workflows are extensively used in the Bioinformatics field. We present a case study that has been implemented in the Italian FIRB LIBI ‘International Laboratory for Bioinformatics’ project [15]. It presents an experiment of multiple sequence alignment (MSA) of human proteins on large scale. In particular the Position Specific Iterative (PSI)-BLAST [14] has been used, a sensitive sequence

similarity search tool, that uses an iterative searching method and a unique scoring scheme to detect weakly related homologues. The goal has been the multiple alignment of a given number of human proteins, stored in the UniProt [2] and TReMBL data banks, against those present in the Uniref90 data bank.

Taking into account that have been considered 1000 proteins of the Uniprot db, grid resources are needed in order to reduce the computational time as well as to automatic several steps, needed for performing the experiment and obtaining the result. PSI-BLAST aligns a protein with those stored in a database by iterating a predefined number of times towards convergence to a solution. This is specified as a parameter input of the application (*j* value).

The PSI-BLAST accuracy depends on the iteration number (*j* value): by increasing the iteration number, the result is better. In this experiment, we chose to set the number of iteration with a value equal to 2 since for these proteins, the algorithm converges to a result.

Data input of the experiment is a flat file data bank, Uniprot (Universal Protein Resource), a comprehensive catalog of information on proteins. From this annotated data bank are extracted in FASTA format the sequences of human protein that they will be used for multiple alignment process. Higher is the score value, lower is E-value and greater is the alignment (the two quantities are inversely proportional). We show the potential of our approach by executing a large parameter sweep (about 2000 executions) as an intermediate step of this workflow in a gLite and Globus based grid environments. In order to support this experiment, several requirements have been met:

- Access to flat file data bank (UniProt NREF Uniref90) with dimension about 800MB;
- Extraction of 1000 sequences by annotated input files (human protein UniProtKB database);
- For each run, the application produces the result of several iterations, specified by the user (two iterations in this experiment). Last iteration is the most important, so output files must be updated;
- Management of produced results that are automatically collected by remote machines;
- Need to reduce the total computing time.

In order to satisfy the access to the data bank, it has been installed on grid nodes, where the application runs, and hence indexed. Indeed PSI-BLAST runs just on indexed data banks. The sequences are extracted by using the above cited library and the parsing of the results allows reducing redundancy. For supporting this experiment, our Workflow Management System has been used. Figure 2 shows the steps related to the workflow, designed using PROGENIA editor.

This workflow is composed by two batch applications (*biolib* and *tar*) and two parameter sweep tasks (*psiblast* and *adjustment*). The complete workflow for a single iteration consists of 5 runs.

The example depicts a workflow including execution, condition and storage vertices; a loop with a fixed number of iterations is also shown. Execution vertices are associated with batch and parameter sweep tasks. Taking into account

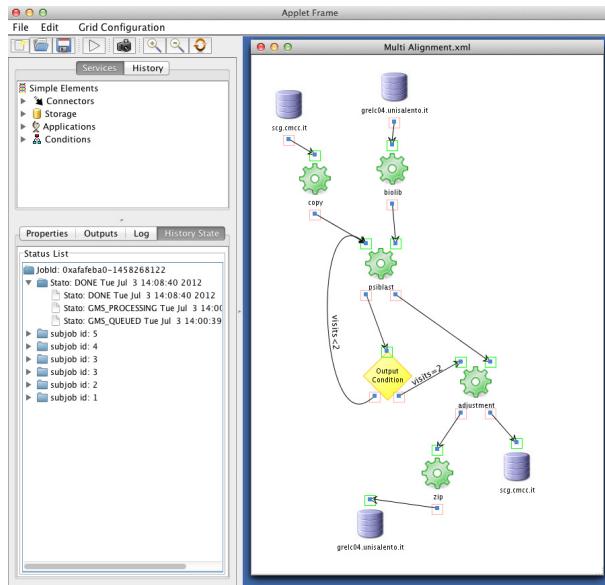


Fig. 2. Workflow for multiple sequence alignment of proteins

that the input data is annotated (in EMBL format), the first step allows extracting each sequence stored in FASTA format (*biolib* task [12]). The input data is stored on a remote machine (*grelc04.unisalento.it* for this experiment). The *biolib* application is responsible to extract a huge amount of sequences from annotated biological data banks. In this phase, a large number of proteins are produced. The output of this run is a directory that contains about 1000 files. Upon *biolib* completion, the user can download the directory through the PROGENIA editor. Each protein in the directory is then aligned using PSI-BLAST. This task will carry out the alignment of computed sequences with those stored in the Uniref90 data bank with an iteration step equal to 1 (indicated with the *j* input parameter of the *psiblast* application). This value, contained in a file, is incremented in the loop cycle of the workflow until the final value is 2 (file based condition). For first iteration of *psiblast* application, the *j* file value is stored in a directory of host *grelc04.unisalento.it* (it is necessary indeed start loop initializing the value of input parameter). The next task (*adjustment* application) starts when *j* equals 2 and all sub-jobs of the parameter sweep task have been executed. *adjustment* application filters each output produced by previous job retaining only the result of the last iteration. So, there are several adjustment processes as the number of *psiblast* sub-jobs. For this reason, this task is also a parameter sweep application. Upon last iteration completion, the result of multi alignment process is compressed (*tar* application) and stage-out on a remote host (*scg.cmcc.it*). Upon workflow submission, the user can retrieve output for each workflow task. We have decided to execute this experiment two times

Table 1. Workflow for MSA Execution time

Workflow Task		Test1 - Globus (time in sec)		Test2 - Globus+WMS gLite (time in sec)		
	j	Time for 1 prot.	Time for 1000 prot.	j	Time for 1 prot.	Time for 1000 prot.
biolib	2		52		2	47
psiblast	1	72	73350	1	324	325600
	2	120	122400	2	360	361444
adjustment	2		2375		12	12178
compress & transfer	2		28		2	25
Total		198 (3.30m)	198205 (55.06h)		700 (11.67m)	699294 (194.25h)

(test 1 and test 2). In the first test, only Globus resources have been used. In test 2, we have executed parameter sweep tasks on gLite resources using a WMS of the BIOMED VO (*egee-wms-01.cnaf.infn.it*), leaving the execution of batch tasks on Globus machines. For parameter sweep tasks, GMS schedules 1000 jobs on each iteration of j , for a total of 2000 runs. In this experiment, all of the results have been collected on a cluster machine using the GridFTP protocol to transfer about 2 GB of data (file sizes range from 280 KB for $j=1$ to 900 KB for $j=2$). The Table 1 describes tests made considering the execution time for each task.

The time measured with the gLite submission in the second test are much higher, because it does not submit directly to resources but through the WMS. Often WMSs are not unreliable because when the job is submitted, its status could be active but the WMS often does not work and the job can remain pending until the expiration time of the proxy and then expires. A solution will be to use CREAM that allows to submit directly to the resources.

4 Conclusion and Future Work

This paper describes PROGENIA WFMS that supports the submission and monitoring of e-Science workflows on distributed grids, such as Globus, gLite and Unicore. The goal of the system has been to provide to e-scientists a software platform where they may define experiments, composing workflows, using a graphical editor, and submitting them on various grids. Future developments of this system is firstly, the implementation of an optimal scheduling policy, in particular for the parameter sweep jobs, that takes into account the fault tolerance mechanisms: given the number of jobs submitted, it is important to foresee a given number of resubmissions, depending on the resource broker of chosen grid middleware. An important aspect will regard the design of a scheduling algorithm that takes into account the status of the grid resources in real time: in order to obtain this result it is needed to design and implement a distributed information service. A P2P solution will be evaluated. Finally, it would be useful to integrate the system within the Cloud Computing platforms.

References

1. Almond, J., Snelling, D.: UNICORE: uniform access to supercomputing as an element of electronic commerce. *Future Generation Computer Systems* 15(5-6), 539–548 (1999)
2. Bairoch, A., Apweiler, R., Wu, C.H., Barker, W.C., Boeckmann, B., Ferro, S., et al.: The Universal Protein Resource (UniProt). *Nucleic Acids Res.* (33), 154–159 (2005), <http://www.uniprot.org>
3. Breton, V., Blanchet, C., Legré, Y., Maigne, L., Montagnat, J.: Grid Technology for Biomedical Applications. In: Daydé, M., Dongarra, J., Hernández, V., Palma, J.M.L.M. (eds.) *VECPAR 2004*. LNCS, vol. 3402, pp. 204–218. Springer, Heidelberg (2005)
4. DEISA. Distributed European Infrastructure for Supercomputing Applications, <http://www.deisa.org/>
5. EGEE. Enabling Grids for e-Science, <http://www.eu-egee.org/>
6. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. *Intl J. Supercomputer Applications* 11(2), 115–128 (1997)
7. GIN-CG. Grid Interoperation Now Community Group, <http://wiki.nesc.ac.uk/read/gin-jobs>
8. Jacq, N., Blanchet, C., Combet, C., Cornillot, E., Duret, L., Kurata, K., Nakamura, H., Silvestre, T., Breton, V.: Parallel Computing, special issue: High-performance parallel biocomputing. *Grid as a Bioinformatics Tool* 30 (2004)
9. Laure, E., Fisher, S., Frohner, A.: Programming the Grid with gLite. *Computational Methods in Science and Technology* 12(1), 33–45 (2006)
10. Mirto, M., Passante, M., Aloisio, G.: A grid meta scheduler for a distributed interoperable workflow management system. In: *Proceedings of the 2010 IEEE 23rd International Symposium on Computer-Based Medical Systems, CBMS 2010*, pp. 138–143. IEEE Computer Society, Washington, DC (2010)
11. Mirto, M., Passante, M., Epicoco, I., Aloisio, G.: An Interoperable Grid Workflow Management System. In: Lin, S.C., Yen, E. (eds.) *Managed Grids and Cloud Systems in the Asia-Pacific Research Community*, pp. 341–357. Springer-Verlag New York Inc. (2010)
12. Mirto, M., Fiore, S., Cafaro, M., Passante, M., Aloisio, G.: A grid-based bioinformatics wrapper for biological databases. In: *Proceedings of the 2008 21st IEEE International Symposium on Computer-Based Medical Systems, CBMS 2008*, pp. 191–196. IEEE Computer Society, Washington, DC (2008)
13. OGF. Open Grid Forum, <http://www.ogf.org/>
14. Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* 17(25), 3389–3402 (1997), <http://www.ncbi.nlm.nih.gov/cgi-bin/BLAST/nph-psi>
15. Italy The LIBI Grid Platform Developers: *Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare*, ch. 29, pp. 577–613, Mario Cannataro, University Magna Graecia of Catanzaro, Italy, Medical Information Science Reference, IGI Global (2009)

OMHI 2012: First International Workshop on On-chip Memory Hierarchies and Interconnects: Organization, Management and Implementation

Julio Sahuquillo, María E. Gómez, and Salvador Petit

Universitat Politècnica de València, Spain

Foreword

Current CMPs include high amounts of on-chip memory storage, organized either as caches or main memory to avoid the huge latencies of accessing off-chip DRAM memory. To address internal data access latencies, a fast on-chip network interconnects the memory hierarchy within the processor chip. As a consequence, performance, area, and power consumption of current chip multiprocessors (CMPs) are highly dominated by the on-chip memory hierarchy and interconnect design. This problem aggravates with the increasing number of cores since a wider and likely deeper on-chip memory hierarchy is required.

Regarding implementation, on-chip cache hierarchies have been typically built employing Static Random Access Memory (SRAM) technology, which is the fastest existing electronic memory technology. However, for current and future technology nodes SRAM presents important design challenges in terms of density and leakage currents as well as manufacturing variation issues, so that it is unlikely the implementation of future cache hierarchies using only SRAM, especially in the context of chip multiprocessors (CMPs). Instead, alternative technologies (e.g. eDRAM or MRAM), which are less prone to failures and address leakage as well as density by design, are being explored in large CMPs.

To take advantage of these complex hierarchies, efficient management is required. This includes, among others, thread allocation policies, cache management strategies, and NoC designs, both in 2D and 3D, involving heterogeneous technologies for memory storage and NoC communications.

The goal of the OMHI workshop is to be a forum for engineers and scientists to address the aforementioned challenges, and to present new ideas for future on-chip memory hierarchies and interconnects focusing on organization, management and implementation.

The contributions of this year nicely reflect the three key points of the workshop's spectrum. The organizers of the workshop would like to thank all the authors for all their very interesting contributions. In this edition, we were able to accept six submissions that were grouped in two sessions. In addition, the organizers were proud to present Professor Ramon Canal as keynote speaker, who gave a interesting talk focusing on the key topics of the workshop entitled "The Memory Hierarchy in the Many-Core Era: Friend or Foe?" which jointly with the paper sessions finally resulted in a nice and very exciting one-day program.

The chairs would like to thank the Euro-Par organizers, the members of the program committee for their reviews and feedback, Ramon Canal and the high number of attendees. Based on the positive feedback of all of them, we plan to continue the OMHI workshop in conjunction with Euro-Par 2013.

Allocating Irregular Partitions in Mesh-Based On-Chip Networks*

Samuel Rodrigo¹, Frank Olaf Sem-Jacobsen¹, and Tor Skeie^{1,2}

¹ Simula Research Laboratory, Norway

² Dept. of Informatics, University of Oslo
`{srodrigo,frankose,tskeie}@simula.no`

Abstract. Modern CMPs require sophisticated resource management in order to provide good utilisation of the chip resources. There exists good allocation algorithms for compute clusters, but these are restricted to specific routing algorithms and not easily transferable to the on-chip domain. We present a novel resource allocation algorithm, TSB, that allows petitions with any shape that is supported by an algorithm implemented using LBDR/uLBDR, and show that this has low complexity and comparable utilisation to UDFlex.

1 Introduction

Chip multiprocessors (CMPs) are becoming as powerful and rich on resources as small computer clusters. Intel's Single Chip Cloud computer (SCC) [1] with 48 cores is a good example of this, and as the name implies, the chip is intended to support multiple concurrent users and applications. With this abundance of resources, efficient management of the on-chip cores is becoming as important as for regular cluster computers.

The efficiency of a CMP to support multiple concurrent applications depends on how well the applications can be allocated resources (cores) on the chip. Some of the key performance points are contiguous versus non-contiguous allocation, performance isolation, and location relative to the memory controllers.

Contiguous allocation means that the application is allocated a set of cores that form a contiguous region, as opposed to non-contiguous allocation where an application can be assigned cores randomly throughout the chip. This impacts the communication bandwidth and latency internal to the application, as well as fragmentation of the on-chip resources. Contiguous allocation gives better application performance, and increases fragmentation since there are strict requirements on which types of partitions that can be allocated.

Performance isolation is a set of different mechanisms that try to make the performance of a single application independent of the state of the rest of the chip. This can be achieved in two ways, using virtual channels or ensuring that the physical network resources are accessible by only a single application. With

* This work has been supported by the project NaNoC (grant agreement no. 248972) which is funded by the European Commission within the Research Programme FP7.

non-contiguous allocation all applications will have to share the physical network resources, so the only option for achieving performance isolation is to assign each application to its own virtual channel. For contiguous regions, however, the routing algorithm can be designed in such a way as to guarantee that all traffic internal to that application remains within the allocated partition. We call this *routing containment*.

Finally, the location relative to the memory controller dictates the cost of accessing the off-chip memory, and also the amount of traffic that will cross through the application partition from other applications accessing the controller.

The effects of the allocation strategies mentioned here have been studied by Triviño et al. [11]. The authors show that contiguous allocation with routing containment gives the overall best performance for the individual applications.

Contiguous allocation with routing containment requires that the application is allocated into logical sub topologies that are supported by the routing algorithm. Current CMP systems rely on XY routing, and routing containment using XY routing requires that the applications are located in sub-meshes. This is very restrictive and can lead to large fragmentation and low chip utilisation. Solheim et al. [9] show that high network utilisation (for clusters) can be achieved by allowing more irregular shapes. However, this requires more flexible routing algorithms to support these shapes. In this case the allocation algorithm presented, UDFlex, relies on Up*/Down* routing.

Recently more sophisticated, yet simple, routing algorithms have been developed for on-chip mesh networks to replace XY. FDOR [8] and LBDR [6] require only a single or a few configuration bits to support a much larger number of different sub-topologies than the simple rectangular meshes supported by XY routing. Evolutions of these simple algorithms such as FJE [7] and uLBDR [6] support a wider range of sub-topologies, even arbitrary shapes, but at an increased implementation complexity.

In this paper we study the improvement in chip utilisation that can be achieved by using these more sophisticated routing algorithms and we devise an allocation algorithm (TSB) that can be efficiently implemented with low complexity and supports any routing implementation from LBDR. The rest of paper is organised as follows. In Section 2 we present some basic resource allocation algorithms from the literature and review studies that concern resource allocation in on-chip systems. We present our novel resource allocation algorithm, TSB, and a minor description of other allocation algorithms in Section 3, and study the efficiency of this algorithm in Section 4. The paper is concluded in Section 5.

2 Related Work

CMP virtualisation (i.e. the ability to efficiently support multiple applications) has been studied in several papers. Flieh et al. [2] studied the concept of CMP virtualisation and identified the key challenges: routing, reconfiguration, and allocation (routing containment, etcetera). The application performance with different application isolation mechanisms has been studied in [11] where the

authors find that full physical performance isolation (using dedicated network resources) gives the best application performance.

In order to exploit the flexibility of the new routing algorithms such as LBDR, an allocated region needs to be configured to provide routing isolation. This requires efficiently configuration mechanisms. Triviño et al. [10] presents an efficient configuration mechanism for the purpose of reconfiguring allocated regions.

Finally, Triviño et al. [12] combined the configuration mechanism and the LBDR routing function with the UDFlex allocation algorithm. The chip is pre-configured with segment-based routing [5], and UDFlex is used to allocate partitions that are contained segments. As we will see in this paper, UDFlex is quite complex in its implementation which can be challenging for short-lived jobs.

The two most common allocation algorithms for mesh networks are First Fit and Best Fit [14]. These algorithms simply search for a rectangular (sub-mesh) area of the desired size and either choose the first one, or the best one which leaves the maximum contiguous region remaining. UDFlex [9] relies on the well-known Up*/Down* routing algorithm to support more flexible partition shapes. The paper shows how this allows for larger system utilisation/less fragmentation while at the same time maintaining good application performance.

3 Allocation Algorithms

3.1 First Fit and Best Fit

First Fit and Best Fit allocation strategies were presented in order to improve the pool of available shapes for allocating an application. Previous strategies only allowed square sub-meshes. To improve the utilisation and minimise the fragmentation, First Fit and Best Fit, allow for rectangular shapes of a requested size $a \times b$, where a can be different to b . Both strategies keep track of the busy and free nodes, and they differentiate in one key strategy. First Fit allocates the job in the first free sub-mesh area. Best Fit, on the other hand, does an exhaustive search to find the smallest free area to allocate the task, in order to reduce external fragmentation. Both of them operate to allocate each task in a contiguous mode. See an example in Figure 1 of both strategies. The scheduler/allocator wants to allocate a 4-cores parallel application. The area composed of nodes 1, 2, 3, 5, 6, and 7 is enough to allocate the job, but the area of nodes 10, 11, 14, and 15 is also valid. Best Fit would choose the second area over the first one.

Notice that while supporting rectangular sub-meshes over square sub-meshes is an improvement to increase chip utilisation, both strategies suffer from internal fragmentation in certain cases by not allowing for more flexible shapes. See an example in Figure 2. As you can see, with XY routing, we want to allocate a 5-core job into a free sub-mesh area of 3×2 area, the only one available. One of the cores, at node 7 for example, will be reserved for the rectangular region, but it will remain unused and unavailable for other applications. XY routing is a very simple mechanism, but cannot cope with irregular shapes, no matter the reason behind the irregularity.



Fig. 1. First Fit and Best Fit example

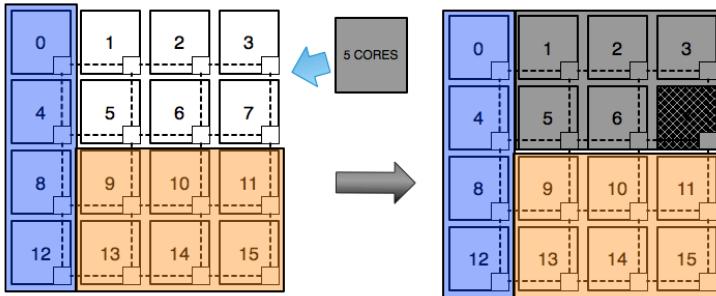


Fig. 2. Internal fragmentation

In the evaluation section, we also present Random Fit. Random Fit is based on the same idea as First Fit and Best Fit, but when allocating a job, instead of choosing the first free sub-mesh area or the smallest one, just performs a pseudo-random choice from the pool of the available ones.

In [14] the complexity of the strategies is presented. First Fit has a complexity of $O(a \times b)$, where a and b are the dimensions of the mesh, and Best Fit displays the same time complexity.

3.2 UDFlex

UDFlex is a tree-based allocation strategy based on the Up*/Down* routing algorithm. In Figure 3 we can see an example of a possible Up*/Down* routing configuration, where node 0 has been assigned as the root of the graph/tree. In order to ensure deadlock-freedom, directions in a link are marked as *up* and *down* (in the figure up is the direction of the arrow), and packets are not allowed to be routed in the up direction after being routed in any down direction.

Once the Up*/Down* routing is defined, UDFlex will allocate the job in a sub-tree graph as long as there is enough free contiguous nodes. Solheim et al. [9]



Fig. 3. Up*/Down* routing on a 4×4 mesh

define two main aims to describe the strategy of UDFlex. UDFlex aims to find the smallest valid Up*/Down* sub-tree graph in a breadth-first search strategy. In the event of a tie, the algorithm chooses the shallowest sub-root relative to the general root of the routing tree to keep that area as free and defragmented as possible. In Figure 4 there are two areas where we can allocate a 4-core job, but the area at nodes 18, 19, 23, and 24, is the one finally chosen, far from the root, which is placed at node 0.

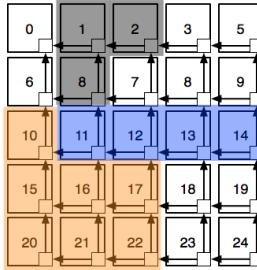


Fig. 4. Placing a 4-core application in UDFlex

UDFlex presents better flexibility for the allocation of different shapes, mostly irregular, to minimise fragmentation. The presence of irregular application domains requires routing mechanisms able to perform on such topologies, like FDOR or LBDR. In order to get better chip utilisation in the design trade-off, we need to aim for routing mechanisms that can display better coverage, coupled with the allocation algorithms. In [9], the authors state that UDFlex presents a complexity of $O(n^2)$, where n is the size of the network.

Although UDFlex allows for different tree formations, depending on which node is configured as a root, it still relies on Up*/Down* routing. One of the problems with this routing algorithm is that most of the traffic tends to focus on the root, which can lead to traffic imbalance and congestion problems. Segment-based Routing (SR) [5], for example, displays better performance than Up*/Down*. We need an allocation algorithm that supports other routing algorithms.

3.3 TSB

In this section we present Tree Segment-Based (TSB), our novel allocation algorithm. TSB is based on the idea of the tree allocation of SBBM [3]. SBBM is a NoC tree-based broadcast routing mechanism that is based on the configuration bits of the LBDR-based routing mechanisms. In SBBM, the switches are able to perform a broadcast operation, ensuring that each one of the nodes receives only one copy of the packet, and they are routed along the allocated tree. The LBDR configuration bits show a flexible routing implementation which can encode different routing algorithms, not only Up*/Down*. The formation of the tree in TSB (as in SBBM) is the result of how these bits are configured, based on the routing restrictions of the routing algorithm, so the designers can aim for different algorithms that display better routing performance or traffic balance, as long as the routing algorithm encoded contains also the property of deadlock-freedom.

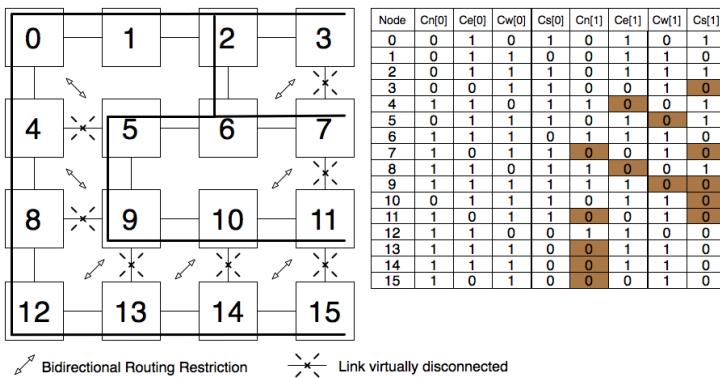


Fig. 5. Forming the TSB tree, change in connectivity bits

In Figure 5 the formation of the tree in TSB is shown. A routing algorithm is encoded in the routing and connectivity configuration bits. The routing bits display information about the routing restrictions and the connectivity bits present the connectivity of the links at each switch. In order to form the tree, the first phase is to produce a copy of the connectivity bits. The algorithm then proceeds to perform a *virtual disconnection pattern* of the links that share the directions of a routing restriction. You can see an example in Figure 6. The rules are simple, proceed to virtually disconnect (reflected by the connectivity bits) one of the links associated by a routing restriction, and in the case that a link is shared by two restrictions, disconnect that one. Although the disconnection can be done randomly, we aim for a balanced wide tree for maximising the bisection bandwidth.

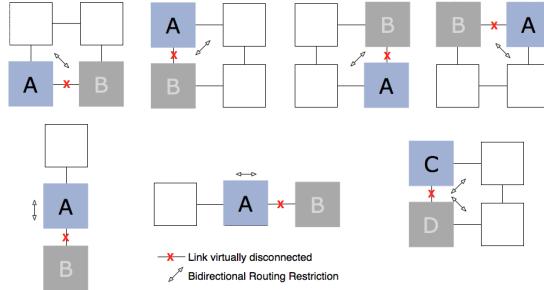


Fig. 6. Disconnection pattern

The second phase of the allocation algorithm, once we have the tree formed, and unlike UDFlex, is to mark the *leaves* of the tree and store them in a list. In order to identify a *leaf* node, the algorithm performs an AND logic on the two sets of connectivity bits on each possible direction (north, east, south and west). If one of the resulting bits is set to one, then we proceed to mark the node as a *leaf*. Notice that the first and second phases are only performed each time the configuration bits are changed, meaning the routing algorithm has changed.

The next phase is to allocate the job choosing from the list of *leaves*. Starting from a *leaf*, the algorithm follows the tree, checking if the nodes are free. If the allocation scheme finds enough resources for the application, then the nodes are marked as busy. As soon as we allocate a job, in order to ensure traffic containment, the connectivity bits in the original layer perform their duty to ensure traffic isolation. The new *leaves* are marked for future jobs, and once the job is deallocated, a change in the connectivity bits of the free nodes starts another *leaf* marking phase. In Figure 7 a tree has been formed upon a given routing algorithm. Nodes 0, 4, 11, 12, 13 and 15 are marked as *leaves*. A job is placed in nodes 10, 13, 14, and 15. Then, node 6 is marked as a new *leaf*.

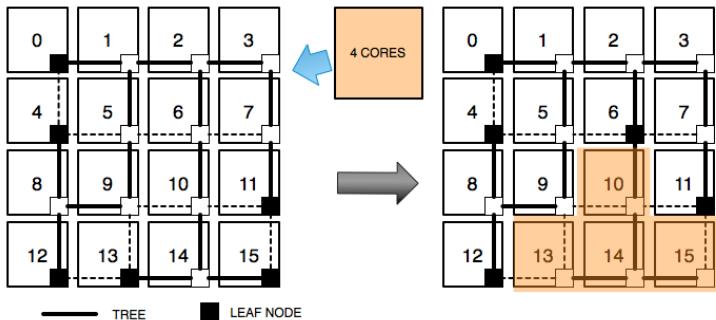


Fig. 7. Result of allocating a job, and new leaves

Notice that in the allocation scheme we are following the tree structure, but we can use another scheme that actually follows the original layer of connectivity bits. This is possible because the formation of the tree is still a subset of the original routing algorithm configuration (the first layer of connectivity bits), and therefore valid for routing purposes inside the application domain, so is up to the designer choose which scheme to follow.

Regarding the complexity of TSB, the main component is searching through the list of *leaf* nodes. As we mentioned before, we aim for a balanced tree as possible. In the conservative case, searching through the list of nodes has a complexity of the *applicationsize* $\times O(\log_2 n)$, which is $O(n \times \log_2 n)$, where n is the size of the network.

4 Evaluation

In this section we describe our evaluation framework. Our simulation framework is based on GEM5 [4], which allows to run a CMP-like scenario with the applications in the SPLASH-2 benchmark [13]. A FCFS (first come, first served) scheduler, which implements the different allocation algorithms, forming a hypervisor is the module who manages the allocation mechanism for the evaluation platform. We performed the evaluations on a 10×10 2-D mesh. The results are averaged from 10 uniform distributions where each application requires randomly 3, 6, 9, or 12 cores for the execution.



Fig. 8. Irregular partitions

In order to reflect the impact of the flexibility of irregular partitions for the allocation task, we evaluated two different scenarios. In the first scenario, we restricted the pool of allowed partitions to rectangular sub-meshes to each one of the allocation algorithms. In the second scenario, we allowed for UDFLEX and TSB to use irregular partitions where FDOR or LBDR can perform routing, like in Figure 8.

In Figure 9 the results for the first scenario are shown. As expected, restricting UDFLEX and TSB to rectangular sub-meshes results in that they perform as well as Best Fit in terms of utilisation. First Fit delivers lower utilisation than Best Fit due to not considering the smallest region to allocate and Random

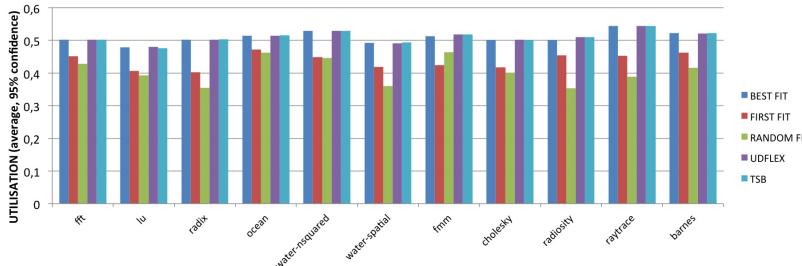


Fig. 9. Rectangular sub-meshes scenario

Fit gets the lowest results, indicating that a smart allocation delivers better performance.

Now, in Figure 10 we see the results for the second scenario. Here UDFLEX and TSB take the advantage of more flexible irregular regions and thus, they minimise the total fragmentation resulting in better utilisation of the chip. They are able to achieve 60% utilisation of the total chip in average.

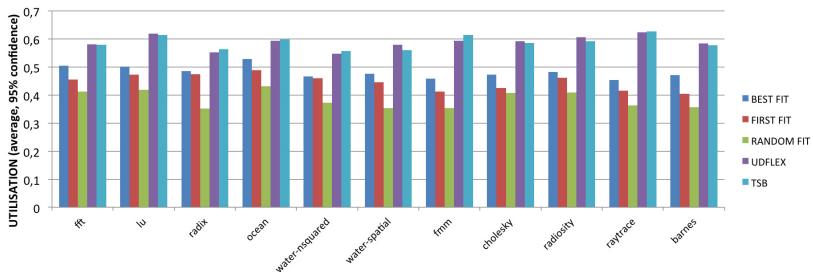


Fig. 10. Scenario with more flexible irregular partitions

5 Conclusion

We have studied the effects of more flexible partitioning when allocating different parallel applications in a CMP. Being restricted to the use of rectangular sub-meshes results in a poor chip utilisation. In this paper, we have contributed with a new idea, TSB, for an allocation algorithm that allows for flexibility in the routing layer, and performs as good as UDFlex, but with support for other routing algorithms.

Future work includes a thorough analysis of more metrics, like queuing and service time of the different allocation algorithms, application performance, overall system utilisation, plus a further evaluation of the impact with near-convex partitions, not just semi-irregular ones.

References

1. Intel Corp. The Single-chip Cloud Computer, <http://techresearch.intel.com/ResearchAreaDetails.aspx?Id=27>
2. Flich, J., Rodrigo, S., Duato, J., Sødring, T., Solheim, A.G., Skeie, T., Lysne, O.: On the potential of noc virtualization for multicore chips. In: International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2008, pp. 801–807. IEEE (2008)
3. Flich, J., Bertozzi, D.: Designing Network On-Chip Architectures in the Nanoscale Era. Chapman & Hall/CRC (2010)
4. Gem5. The gem5 simulator system, <http://gem5.org>
5. Mejía, A., Flich, J., Duato, J., Reinemo, S.A., Skeie, T.: Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori. In: International Parallel and Distributed Processing Symposium, p. 84 (2006)
6. Rodrigo, S., Flich, J., Roca, A., Medardoni, S., Bertozzi, D., Camacho, J., Silla, F., Duato, J.: Addressing manufacturing challenges with cost-efficient fault tolerant routing. In: NOCS 2010: Proceedings of the 4th ACM/IEEE International Symposium on Networks-on-Chip, pp. 25–32 (2010)
7. Sem-Jacobsen, F.O., Rodrigo, S., Skeie, T., Strano, A., Bertozzi, D.: An efficient, low-cost routing framework for convex mesh partitions to support virtualisation. ACM TECS Special Issue on OnChip and OffChip Network Architectures (2011)
8. Skeie, T., Sem-Jacobsen, F.O., Rodrigo, S., Flich, J., Bertozzi, D., Simone, M.: Flexible DOR Routing for Virtualization of Multicore Chips. In: International Symposium on System-on-Chip (2009)
9. Solheim, Å.G., Lysne, O., Sødring, T., Skeie, T., Libak, J.A.: Routing-Contained Virtualization Based on Up*/Down* Forwarding. In: Aluru, S., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2007. LNCS, vol. 4873, pp. 500–513. Springer, Heidelberg (2007)
10. Triviño, F., Alfaro, F.J., Flich, J., et al.: NoC Reconfiguration for CMP Virtualization. In: 2011 10th IEEE International Symposium on Network Computing and Applications, NCA, pp. 219–222. IEEE (2011)
11. Triviño, F., Sánchez, J.L., Alfaro, F.J., Flich, J.: Virtualizing network-on-chip resources in chip-multiprocessors. Microprocessors and Microsystems, 1–16 (October 2010)
12. Triviño, F., Sánchez, J.L., Alfaro, F.J., Flich, J.: Exploring noc virtualization alternatives in cmps. In: 2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, pp. 473–482. IEEE (2012)
13. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 programs: characterization and methodological considerations. In: ISCA 1995: Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp. 24–36. ACM, New York (1995)
14. Zhu, Y.: Efficient processor allocation strategies for mesh-connected parallel computers. Journal of Parallel and Distributed Computing 16, 328–337 (1992)

Detecting Sharing Patterns in Industrial Parallel Applications for Embedded Heterogeneous Multicore Systems

Albert Esteve, María Soler,
Maria Engracia Gómez, Antonio Robles, and José Flich

Universitat Politècnica de València,
Camino de Vera, s/n, 46022 Valencia, Spain
[{alesgar,masohe}@gap.upv.es,](mailto:{alesgar,masohe}@gap.upv.es)
{megomez,arobles,jflich}@disca.upv.es
<http://www.upv.es/>

Abstract. Embedded devices are becoming more and more present everywhere. Moreover, mobile devices are becoming also more computationally powerful. These embedded architectures present new challenges since they execute several applications that must preserve security, allow sharing information in a coherent way, to be scalable and provide the required levels of performance, while at the same time they must be power efficient. The vIrtical project focuses on these challenges.

In this context, as a starting point, we tackle the characterization of applications targeted for the hardware platform developed, that is, a heterogeneous multicore SoC. The aim is to analyze memory sharing patterns in order to exploit them to make the coherence protocols more scalable and power-efficient.

We have identified that 60% of the accessed blocks are data, and from those only 40% require coherence maintenance.

Keywords: sharing patterns, embedded systems, cache coherence.

1 Introduction

The use of computers has been extended to most areas of our everyday life. These embedded devices present new challenges of security, scalability and power efficiency. In the vIrtical project we focus on these challenges and as a part of this, we address the design of the coherence protocol to be adopted to the need of these heterogeneous multicore platforms.

The cache coherence problem arises when copies of the data stored at several private caches associated to different cores, are reachable at the same time by two or more cores and modifiable by some of them. The cache coherence protocol must guarantee coherence of the data through the entire system which means deciding how and when a single core is granted permission to modify data and ensuring that subsequent readings of the written data by other cores will attain updated copies of the modified data (multiple readers). To do so, different cache

coherence mechanisms can be applied. Commonly, these mechanisms introduce certain overhead in terms of either coherence traffic issued and storage resources required, which can significantly penalize performance, increasing the execution time of the running applications as well as power consumption. The current trend to increase the number of cores into CMP and MPSoC systems further aggravates this problem, as the cache coherence protocol does not scale due to its resource overheads and its indirection when accessing data (access latency is increased).

On the other hand, a different approach to tackle the coherence problem has recently been proposed ([1], [2] and [3]), consisting on removing coherence maintenance for those data objects that do not need it, either because they are not shared (private to one core) or because they are shared but never written by any core. This approach requires the use of effective mechanisms to identify data blocks that do not need coherence maintenance, which in turn may introduce certain overhead. However, the success and suitability of the selected coherence mechanism will strongly depend on both the architectural context of the system it is being applied to and the sharing patterns of the applications running on the system.

Therefore, a detailed analysis of the sharing patterns of the applications to be supported is needed, in order to identify opportunities of applying one or another cache coherence mechanism together with different coherence optimization techniques.

The paper is organized as follows. Section 2 gives an overview of the capturing methodology used to obtain the data analyzed. Section 3 focuses on sharing patterns, detailing the main aim of the study and showing our analysis results. Finally, in Section 4 some conclusions derived of the previous analysis are given.

2 Analysis Methodology

This section describes the target system used for the analysis and the methodology followed in order to capture the information required to properly perform it.

2.1 Target System

We assume an ARM quad-core Cortex-A15 MPCore as the system processor. The characteristics of the memory hierarchy simulated are:

- 4 L1 caches (one per core) have 128 sets, 4 ways and a line size of 64 bytes.
- 1 L2 cache (shared by all the cores) has 512 sets, 16 ways and a line size of 64 bytes. Caches are inclusive (L1 caches' content is included in L2).

2.2 Simulation Tools

ARM FastModels simulator provides out of the box programmer's view models of the ARM processors. Thus, it is both functionally accurate and easy to use

since ARM processors models are already implemented as an Instruction Set Simulator. We use this simulator to model the target system and to run the targeted applications on top of it.

The model of ARM Cortex-A15 provided with FastModels is capable of running basic applications, but it does not cover all the requirements of an operating system, which is needed to evaluate and benchmark parallel applications. We thus use a more complex model also provided with FastModels (namely RTSM-VE Cortex-A15) that allows the simulation of both operating systems and applications. In this RTSM-VE model the cores are connected directly to a Versatile Express platform through a 64-bits AXI bus. This platform includes the Motherboard Express μ ATX, which has been especially designed to support future generations of ARM processors, and the CoreTile Express daughterboard with the on-board DDR2 SDRAM.

2.3 Trace Acquisition Methodology

In order to perform the sharing pattern analysis we need to capture all memory accesses from the cores, being our interval of interest the parallel section of the applications, namely the section executed by several cores at the same time. To identify this section we explored the application code trying to reach the starting point and end point of threads. To delimit this section and make it recognizable by FastModels we introduce a special *nop* instruction on the application available on the ARM Instruction Set.

FastModels supports the use of a Model Trace Interface (MTI) plug-in that permits us to consistently track the execution of the model. Through implementing an MTI plug-in for tracing memory accesses produced by the cores and adding it to the simulation we are able to trace exactly what we need in the form that we require.

MTI plug-in provides many different sources to trace, but the more verbose the trace obtained is and the more sources are involved, the more it slows down the simulation. Since it takes billions of instructions to boot a Linux system on FastModels, we need to deactivate the output and minimize the number of sources of the tracing until the starting point of the segment of interest is detected. We have achieved an acceptable compromise solution by capturing only the instructions fetched by the cores until we reach the aforementioned special *nop*, and subsequently tracing loads, stores, and fetches until we get to the ending special *nop*.

The ARM Simulator provides programmer's view models with some limitations. On system simulators there is a trade-off between speed and accuracy. FastModels in particular opts for the execution speed thus lacking some features needed for our analysis, such as:

- Instruction timing: a processor issues a set of instructions (a.k.a a quantum) at the same point of the simulation time, and then waits some amount of time before executing the next quantum, being impossible to determine the right time each individual instruction is executed.

- Bus traffic: bus traffic has several optimizations that make it inaccurate.
- It does not support out-of-order execution and write-buffers as architecturally defined: execution on FastModels is only an approximation to execution of architecture and it must be thus considered.

2.4 Applications

We have obtained traces with five of the many different algorithms in the OpenSSL suite, three hash algorithms (SHA1, SHA256 and SHA512), and two encryption algorithms (AES-128-ECB and AES-256-ECB).

3 Sharing Patterns Analysis

As commented above, there exist several recent proposals that take advantage of memory block classification for different purposes, such as enhancing efficiency of directory caches, reducing coherence overhead or better taking advantage of NUCA caches. All of them are mainly based on the classification of blocks in private (P) and shared (S). Moreover, some others extend this classification to read (R) only and written (W).

So the scheme we propose in order to analyze blocks is made classifying them as:

- PR (Private Read-only): Only one processor accesses the block. All accesses are loads.
- PW (Private read-Write): Only one processor accesses the block. At least one access is a store.
- SR (Shared Read-only): At least two processors access the block. All accesses are loads.
- SW (Shared read-Write): At least two processors access the block. At least one access is a store.

According this classification, the only blocks that actually need coherence maintenance are the SW ones and therefore we can take advantage of the fact that the remaining blocks do not need it, either because they are accessed by just one core or because they are only read by any number of cores. So special attention will be paid to SW blocks.

The classification schemes proposed in the literature have used different granularities: blocks ([4] and [5]) and pages (OS-based schemes, as can be seen in [1], [2] and [3]), looking for a trade-off between detection accuracy and the required overhead. So our analysis is made with three different granularities based on blocks and pages: 64 bytes block, 4 Kbytes pages, and 64 Kbytes pages. This is interesting since coherency with page granularity is easier to implement and manage. Working at page level allows us to rely on the operating system to detect whether coherency needs to be applied or not, aiding to reduce the hardware

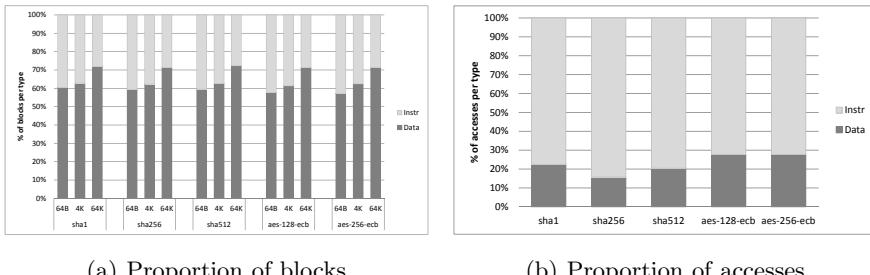
overhead and complexity. On the other hand, the use of page level granularity allows us to analyze how critical is the block misclassification introduced with coarser grains. In addition, studies at page level granularity are intended to identify the viability of applying cache coherency at page level instead of block level.

Basically, we provide two kinds of analysis. The first one, referred to as static analysis, is intended to count the number of blocks included in each category. The second one, referred to as dynamic analysis, shows the number of accesses to blocks for each category. Both views complement each other and allow us to identify which categories of blocks are the most frequent and which ones are the most accessed.

The previous classification will help us later to identify which are the best optimization opportunities when developing the appropriate coherence protocol.

3.1 Analysis Results

The results presented are focused on the data blocks or pages. As can be seen on the Fig. 1(a) and Fig. 1(b) the proportion of data blocks and accesses is more relevant. Also, due to the fact that most of the instruction blocks are shared through all four cores, the classification between Private and Shared instruction blocks is less interesting in this case. Finally, we detected no interleaving between data and instruction blocks at different page granularities.



(a) Proportion of blocks

(b) Proportion of accesses

Fig. 1. Data/Instruction classification

In all figures, the results obtained for each of the analyzed applications, together with the resulting average values, are displayed.

Fig. 2(a) shows the block classification based on the detection of the Private-Shared Read-Write scheme for every block requested at different granularities. First of all, it is observed that, on average, 40% of data blocks are private (PR or PW). The remaining blocks are shared (60%), but notice that indeed only 40% of data blocks are SW, that is, they require coherence maintenance. However, this promising result vanishes when the granularity used for classifying the blocks is increased. As can be observed, the coarser grain used, the more shared and

written blocks are found. In particular, for 4KB pages, the percentage of SW blocks is greater than 60%, whereas this percentage, on average, exceeds the 80% for 64KB pages. This means that SW blocks are concentrated in a certain number of pages, but they are mostly distributed among them. As a consequence, the detection accuracy decreases as far as the granularity is increased in order to simplify the detection process, leading to a misclassification of page blocks. Notice that just one SW block contained in a page will cause the page to be classified as SW.

Fig. 2(b) shows the dynamic analysis. It is observed that despite the fact that SW blocks just represent 40% of the total number of blocks, as was shown above, they agglutinate tough the larger number of accesses (60% on average). Furthermore, the number of accesses to private data blocks is indeed negligible. Unlike the static analysis, the larger differences between applications are observed here. Also, the number of accesses classified as SW hardly increases as granularity becomes coarser. On average, it reaches a 70% for 64KB pages. Notice that this is an expected result as long as the highest percentage of accesses inside a page is destined to SW blocks. Given that most of data memory accesses require coherence maintenance, the design of the cache coherence strategy will be a key element to provide high performance.

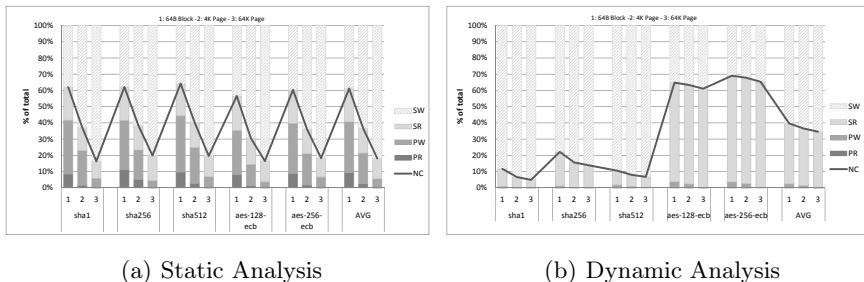
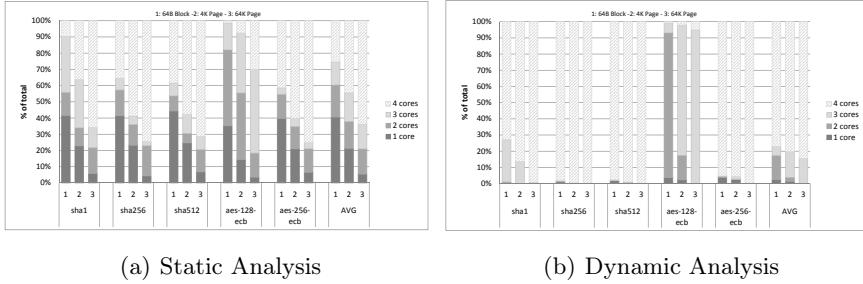


Fig. 2. Static and dynamic Analysis per block type

In order to offer a deeper insight into the sharing degree of data blocks, let us analyze to what extent they are shared, that is, how many cores share each of these data blocks. So, in Fig. 3.1 we analyze the number of sharers per block. If there are no sharers and the block is only accessed by one core, it corresponds with a Private block detected in the previous analysis. We consider also both static (Fig. 3(a)) and dynamic analysis (Fig. 3(b)). In Fig. 3(a), we can observe how, on average, about 20% of the blocks are shared by just two cores, 15% are shared by three cores, and 25% of them are shared by four cores. As the detection granularity increases, the number of blocks shared by all the cores is larger. The reason is the same as that pointed out with respect to Fig. 2(a). Moreover, from Fig. 3(b), it is observed that the most accessed data blocks are those shared by all four cores. This result corroborates even more the importance of carrying



(a) Static Analysis

(b) Dynamic Analysis

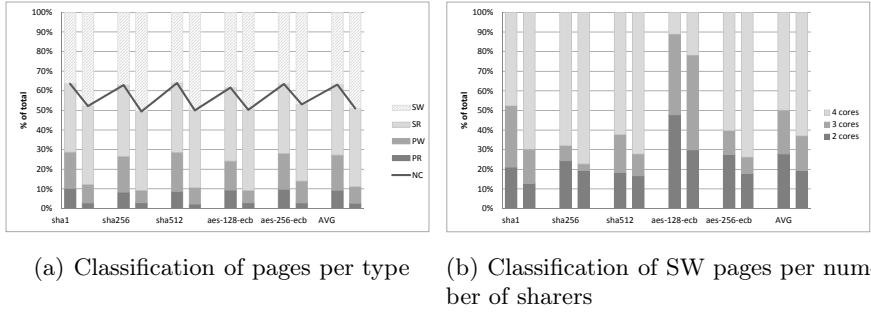
Fig. 3. Static and dynamic Analysis per number of sharers

out an appropriate design of the cache coherence mechanism as far as a large number of cores are usually involved in the coherence maintenance of the data blocks.

Until now, the analysis has been focused on classifying blocks by using different detection granularities. However, it is also interesting to just classify pages. It may be important to assess the convenience of managing cache coherence in a per page basis instead of the usual strategies based on block tracking. In this case, page classification in PR, PW, SR, and SW classes is as follows. A page is classified as SW when it contains at least a SW block. Otherwise, it will be classified as SR if at least one of their blocks is SR. On the contrary, if the page does not contain neither SW nor SR blocks, it will be classified as PW if at least it contains a PW block. Otherwise, the page will be classified as PR. In this sense, Fig. 4(a) shows the page classification for 4KB and 64KB page sizes. As can be seen, more than 35% of the 4KB pages require coherence (they are SW), whereas this percentage increases until near 50% for 64KB pages. This means that SW blocks are not spread over all the pages, but they are indeed distributed between a limited number of pages, larger as the page size increases.

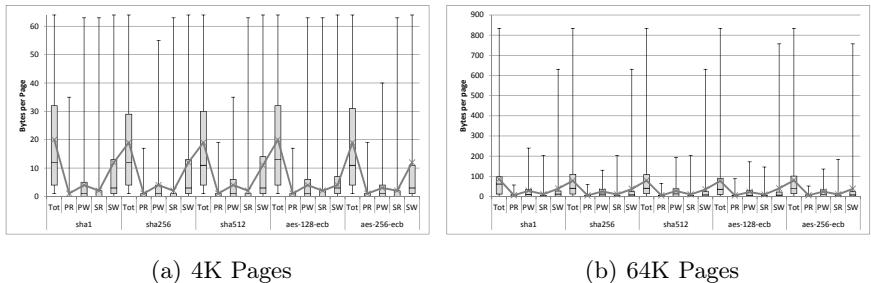
In order to offer a deeper insight into SW pages, from now on, SW pages become the focus of our analysis. Firstly in Fig. 4(a) we classify pages per access type and subsequently in Fig. 4(b) we discern the number of sharers on pages classified as SW. As can be seen, like it was observed on analyzing block sharing, most pages are shared among three or more cores, as more as larger the page size is. In particular, half of the blocks are shared between 4 cores for 4KB pages, whereas the percentage exceeds the 60% when page size is 64KB.

As commented before, the main disadvantage of using page granularity to detect blocks requiring coherence is the misclassification that page blocks may suffer. Notice that a single block can determine the classification of the rest on the same page. In order to analyze this effect, we study in depth the internal anatomy of SW pages. The study is performed by considering the type each block in the page had been assigned in case of having assumed block granularity in the detection process. In particular, we proceed to count the number of each of the block types contained in the page. Information is represented making use

**Fig. 4.** Page Classification

of box-and-whiskers plots. This will help to determine how populated the pages are and the real significance of block misclassification.

Fig. 5(a) and Fig. 5(b) show results for 4KB and 64KB page sizes, respectively. First of all, it is observed that pages are hardly populated, so much less, the larger the page size is. However, a great variability is observed, from pages hardly containing a few blocks until pages crowded with blocks. On average, the medium value of blocks per page is about 20 out of 64 for 4KB pages and 80 out of 1024 for 64KB pages. Anyway, the precise distribution of the total number of blocks and the number of blocks of each type can be observed in the aforementioned figures. Regarding SW blocks, it can be observed that, for a page size of 4KB, most of the blocks in SW pages are SW blocks, but when the page size is increased, the PW blocks become more frequent. Despite this, the number of SW blocks present in the page is very small in relative terms (on average, the 75% of 4KB pages have less than 12 SW blocks, whereas the 75% of 64KB pages have less than 25 SW blocks). These results may suggest the possibility of applying fine grain detection techniques inside SW pages in order to isolate true SW blocks, thus limiting coherence maintenance actions to them.

**Fig. 5.** Proportion of blocks per type on SW pages

We have also performed a dynamic analysis of SW pages in order to obtain the proportion of store access. As can be seen in Fig. 6, about 30% of the memory accesses to SW pages are stores. Obviously, these accesses will be destined to either SW or PW blocks. This kind of studies allow us to assess the convenience of applying update strategies instead of invalidate ones.

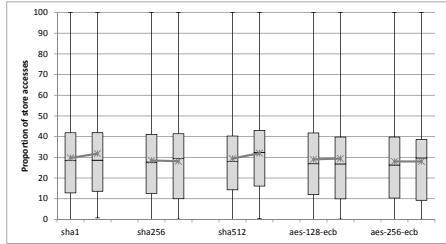


Fig. 6. Proportion of store accesses on SW pages

Finally, it is carried out an analysis in search of the existence of producer-consumer patterns in SW pages. Fig. 7(a) shows the proportion of blocks presenting the producer-consumer pattern related to the amount of blocks requested within the page for different page granularities. As observed this percentage is relatively small. So, in order to definitively observe whether or not the producer-consumer pattern is relevant on the analysis, we studied the percentage of accesses done to blocks presenting the pattern within the SW pages. As can be seen on Fig. 7(b) not even the 0.003% of the accesses are done to these blocks.

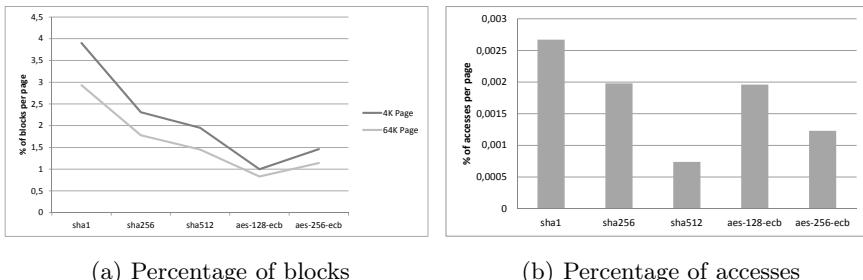


Fig. 7. Proportion of blocks with producer-consumer pattern on pages SW

4 Conclusions

On average, about 60% of the blocks accessed by the analyzed applications correspond to data blocks, which, unlike instruction blocks, may require coherence

maintenance. Moreover, despite the fact that it is detected a high sharing degree of data blocks among cores, indeed only SW blocks (just 40% of the total number of data blocks) require coherence. However, SW blocks agglutinate the largest number of accesses (60% on average). That means that the majority of data accesses require coherence. Therefore, its impact in performance can be significant.

Also, it is observed that SW blocks are not spread over all the pages, but they are indeed distributed between a limited number of pages. In particular, less than 40% of data pages require coherence. Deeping inside SW pages, we observe that they hardly are populated, containing about 30% and 10% of blocks, on average, for 4KB and 64KB pages, respectively. Among them, the number of SW blocks is the majority. Furthermore, about 30% of the accesses to blocks of SW pages correspond to store operations. Therefore, a fine grain detection inside SW pages may be interesting, at the expense of introducing additional hardware support.

Acknowledgements. This work has been supported by the VIRTUAL project (grant agreement n° 288574) which is funded by the European Commission within the Research Programme FP7. Eight partners are involved in this project (Universitat Politècnica de València, Università di Bologna, STMicroelectronics, Thales, Technological Educational Institute of Crete, SYSGO, ARM and VOSYS).

References

1. Cuesta, B., Ros, A., Gómez, M.E., Robles, A., Duato, J.: Increasing the Effectiveness of Directory Caches by Deactivating Coherence for Private Memory Blocks. In: 38th Int'l Symp. on Computer Architecture, ISCA, pp. 93–104 (June 2011)
2. Hardavellas, N., Ferdman, M., Falsa, B., Ailamaki, A.: Reactive NUCA: Near-optimal block placement and replication in distributed caches. In: 36th Int'l Symp. on Computer Architecture, ISCA, pp. 184–195 (June 2009)
3. Kim, D., Ahn, J., Kim, J., Huh, J.: Subspace snooping: Filtering snoops with operating system support. In: 19th Int'l Conference on Parallel Architectures and Compilation Techniques, PACT, pp. 111–122 (September 2010)
4. Hossain, H., Dwarkadas, S., Huang, M.C.: POPS: Coherence protocol optimization for both private and shared data. In: 20th Int'l Conference on Parallel Architectures and Compilation Techniques, PACT (October 2011)
5. Pugsley, S.H., Spjut, J.B., Nellans, D.W., Balasubramonian, R.: SWEL: Hardware cache coherence protocols to map shared data onto shared caches. In: 19th Int'l Conference on Parallel Architectures and Compilation Techniques, PACT, pp. 465–476 (September 2010)

Addressing Link Degradation in NoC-Based ULSI Designs

Carles Hernández², Federico Silla¹, and José Duato¹

¹ Universitat Politècnica de València
DISCA, Camino De Vera s/n, 46022 Valencia, Spain
fsilla@disca.upv.es

² Barcelona Supercomputing Center
Carrer de Jordi Girona 29, 08034 Barcelona, Spain
carles.hernandez@bsc.es

Abstract. Process variability makes silicon devices to become increasingly less predictable, forcing chip designers to create different techniques to avoid losing performance and keeping yield. NoC links are also affected from process variation. Actually, the probability of having faulty links in a NoC might considerably increase in future CMP systems, expected to be implemented with 22nm technology by 2015.

In this paper we propose a new technique to overcome the presence of failures in NoC links. The proposed mechanism, a variable pitch-size NoC architecture, is intended to face both manufacturing defects and variation-induced timing errors. Our new mechanism adapts link operation to the real conditions of the manufactured chip and therefore it is able to keep links working in the presence of variations.

Simulation results show that most of the still available bandwidth present in links affected by process variation can be retrieved, thus avoiding the performance degradation that other mechanisms, like reducing link frequency, would introduce.

Keywords: process variation, NoC-based CMPs, ULSI designs.

1 Introduction

Process variability makes silicon devices to become increasingly less predictable, forcing chip designers to create different techniques to avoid losing performance and keeping yield. The immediate technique used to guarantee the proper operation of the chip against variation-induced timing failures is reducing clock frequency, so that all the parts of the chip can properly work. Unfortunately, this low-cost technique is not useful as variability increases because of the large performance penalty. Additionally, according to the predictions of the ITRS [4], defect density levels also increase with technology scaling. In this regard, NoC links are specially prone to manufacturing defects, as they are usually routed in the upper metalization layers and require a high number of vias to reach active devices located at the silicon surface [5]. Actually, the probability of having faulty

links in a NoC might considerably increase in future CMP systems, expected to be implemented with 22nm technology by 2015 [9], due to the great variability caused by the much smaller transistor size, the increase of defect density levels, and the huge number of links present in the on-chip network.

Therefore, new fault-tolerant link designs are required to deal with the presence of failures. In this paper we propose a new technique to overcome the presence of failures in NoC links. The proposed mechanism, a variable phit-size NoC architecture, is intended to face both manufacturing defects and variation-induced timing errors. Our new mechanism adapts link operation to the real conditions of the manufactured chip and therefore it is able to keep links working in the presence of variations. The benefits of such a technique are twofold. On one hand, such a variability-aware mechanism avoids chip performance to be significantly decreased. On the other hand, yield is maintained. Otherwise, an important fraction of the manufactured chips should be discarded.

2 Related work

Recently, variability-aware design has arisen as one of the hot topics in computer architecture and high speed digital design. Consequently, the impact of variations in circuit performance has been thoroughly analyzed. Several works in the literature pointed out the importance of random process variations in NoC interconnects. In this sense, the implications of variations in NoC link interconnect are analyzed in [6][10]. Concretely, in [6] different measures of delay uncertainty for technologies from 45nm to 16nm are provided. Similarly, the authors of [10] analyze delay variation of next technology nodes. Both studies remark the importance of process variation in communication links when technologies scale down.

There are several proposals that are able to tolerate the presence of faulty wires. Some of them are based on tolerating infrequent run-time timing violations, where delay failures are tolerated at the cost of performance because errors involve re-transmission [3][8]. On the contrary, other kind of proposals focus on increasing interconnect yield by using redundant hardware. In this sense, the authors of [5] propose the use of spare wires to tolerate a bounded number of faults without decreasing communication performance.

3 Problem Statement

As mentioned before, our goal is to deal with links where not all wires are fault-free, caused either by delay variations or by the presence of manufacturing defects.

Delay uncertainty makes wires in a link to present a maximum achievable frequency lower than the design frequency. An example of such a link is shown in Figure 1. Figure 1(a) shows a link composed of five wires working at the frequency targeted at design time, f_{clk} . This is a non-faulty link. Figure 1(b) shows a faulty link. In this link, wires 1 and 4 are not able to switch at the

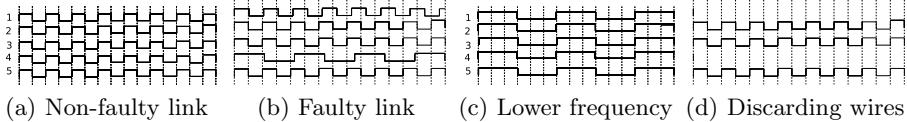


Fig. 1. Example of a 5-wire link under variation-induced timing failures and two ways of keeping it working

original f_{clk} frequency. Wire 1 is slightly slower than the design frequency (90% of f_{clk} , approximately) while wire 4 switches at less than half of the initial frequency (40% of f_{clk}). The link in Figure 1(b) would usually be labeled as a faulty link at system initialization so that its use is precluded. In fact, if the original link shown in Figure 1(a) has an aggregated bandwidth $B_a = 5bw$ bps, where bw is the targeted bandwidth of each wire, the faulty link in Figure 1(b) is still able to deliver approximately $B_b = 3bw + 0.4bw + 0.9bw = 4.3bw$ bps representing 85% of the initial bandwidth. In real NoC links composed of 128 or 256 wires, for example, having a few slower wires would mean that almost 100% of the bandwidth is still available. Thus, discarding the entire link means wasting bandwidth.

There are multiple possibilities to retrieve bandwidth from faulty links. When failures are caused by process variations the easiest solution is reducing the frequency of the link to operate at the frequency of the slowest wire [11]. This option allows the link to be operational at the expense of a considerable reduction in performance. In the case of the 5-wire link shown in Figure 1(c), the available bandwidth retrieved with this technique would be $B_c = 0.33(5bw) = 1.65bw$ bps. Note that this important reduction in bandwidth would also be present in real and wider links, because the whole link reduces its frequency independently of the number of wires.

On the contrary, when link malfunction is caused by manufacturing defects, like shorts and opens, reducing the frequency of the link is not enough to keep the link working. In those cases, the immediate approach to tolerate failures is by precluding the use of faulty links. For that purpose, fault-tolerant routing mechanisms are required [1]. However, avoiding the use of some faulty links may considerably reduce bisection bandwidth causing an important reduction of network performance.

In this paper we propose a different approach to retrieve the bandwidth still available. This technique is based on discarding the wires that are faulty regardless of the origin of failures. This idea is shown in Figure 1(d), where wires 1 and 4 are not used. In this case, link bandwidth is reduced to $B_d = 3bw$ bps, that corresponds to 60% of the original bandwidth. When real links with 128 or 256 wires are considered, between 90% and 95% of the initial bandwidth is still available. Obviously, in order to properly transmit information, flits must be suitably sliced and their bits sent across the non-faulty wires. Therefore, additional hardware is required at the transmitter to slice flits according to the available wires and to

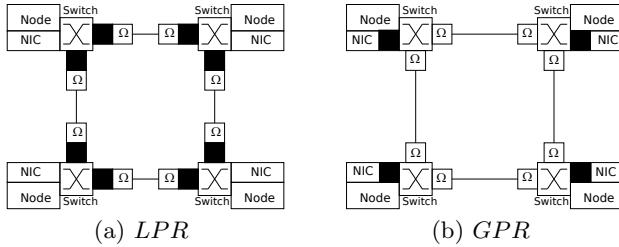


Fig. 2. Diagram of the proposed variable phit size NoC architecture

allow the transmission of bits belonging to two consecutive flits during the same clock cycle, if necessary. Also, additional hardware is required at the receiver to retrieve the original flits. This technique is called *Phit Reduction*. We will refer to it as *PR*.

4 Variable Phit-Size NoC Architecture

In order to implement our proposal, two possibilities arise. The first one, hereafter named *Local Phit Reduction* (LPR), is oriented to fabrication processes with very high variability or a high density of manufacturing defects, because, in that case the differences in the number of faulty wires among different links are high. This approach requires the inclusion of complex modules in every router port. The included port modules that enable the use of the LPR approach must be able to slice flits and to transmit them across the non-faulty wires, and later reconstruct them. Thus, this mechanism is costly in hardware but provides good performance because transmission across each link is performed using the maximum available link bandwidth.

The other way of implementing this technique is named *Global Phit Reduction* (GPR). When differences in the expected number of faulty wires across the links are low, the inclusion of a hardware module to slice and rebuild flits at every port of the network wastes, unnecessarily, silicon area. In this case, reducing the phit size for the whole network is a better solution. In order to perform this reduction, it is necessary to identify the link with more faulty wires. To do so, an initialization algorithm is run across the network to find such a link. Once this link is found, it will bound the phit size for the whole network to the number of its non-faulty wires. With this approach, the number of hardware modules needed is considerably reduced as they are only required at the elements able of injecting and extracting traffic to and from the network.

As can be seen, the variable phit-size NoC architecture proposed in this chapter consists on the use of receiver and transmitter modules to deal with flits with a variable phit size. Figure 2 shows the arrangement of the required modules for applying both the LPR and the GPR mechanisms in a 4-node network. In Figure 2, black squares represent both transmitter and receiver modules able to inject flits with a variable phit size, thus including the logic required to slice

and rebuild flits. On the other hand, boxes labeled as Ω represent NxN 1-bit wide omega crossbars, being N the initial link width. Figure 2(a) shows that when using the *LPR* approach, the complex hardware modules are required at every external output/input port. On the contrary, when using the *GPR* approach (Fig 2(b)) the complex hardware transmitter and receiver modules are required only at network interfaces (NIC) as all transmissions are based on a given phit size used in the whole network (or region)¹. Finally, to select a subset of non-faulty wires among all the total wires of a link, Ω link-crossbars are placed in between the router data path and the physical link in both output and input links.

5 Evaluation

In this section the variable phit-size NoC architecture is evaluated. First, we focus on the details of 2D NoC link designs. Later, the performance and area overheads of the proposal are analyzed.

5.1 NoC Link Model

Repeater insertion is an efficient method to reduce interconnect delay and signal transition times. However, in order to design links with the minimum possible power dissipation, a different approach is required. For that purpose, we follow the reasoning given in [2]. Concretely, links are designed using the minimum device size that allows to reach a given target frequency. In this study we consider two different link designs. The first link, the High-Performance link (*HP*), is designed for working at 3GHz. The second link, the Low-Power link (*LP*), is designed for achieve a working frequency of 2GHz. In both cases we consider a link length of 2.4mm, according to the CMP floorplan designed in [7]. As this floorplan refers to a 45nm CMP implementation, link length has been scaled for the rest of technologies considered in this study. Table 1 summarizes link configuration for both the *HP* and the *LP* scenario. Notice that a given link configuration is determined by the number of sections (k) and the size of its repeaters (h).

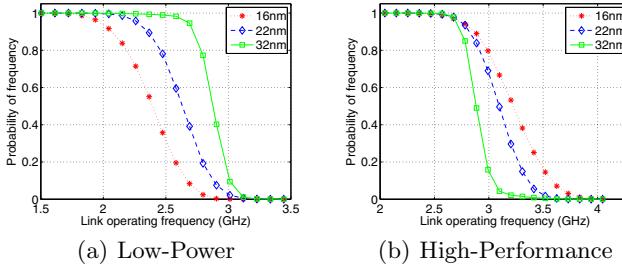
5.2 Injecting Process Variations into NoC Links

Before presenting a performance comparison of our proposal it is necessary to present how process variation affects links. In order to measure the performance of the proposed *PR* mechanism against process variations, we have injected parameter variations in all link wires of 100 chips instances using the framework presented in [7]. For the placement of links we used the 64-core floorplan also

¹ Note that regions with different phit size can be built using the proposed architecture in order to improve the behaviour of the design in the presence of highly spatially correlated variations.

Table 1. Link configurations used in the study

Technology	Length(mm)	High Speed		Low Speed	
		delay(ns)	k h	delay(ns)	k h
32nm	1.71	0.306	5 18	0.48	3 10
22nm	1.17	0.328	5 16	0.45	3 10
16nm	0.85	0.33	5 16	0.44	3 10

**Fig. 3.** Impact of variation in wire delay for both high-performance and low-power links

presented in [7]. Links considered for this study are 128-bit wide. The variability sources injected are transistor effective channel length variation ($\sigma_{L_{eff}}$), random threshold voltage variations ($\sigma_{V_{th}}$), and metal thickness variation (σ_{m_t}). It is important to remark that the severity of variations applied to the designed links is the one provided in the ITRS report as expected for the technologies considered in this study.

Figure 3 shows the impact of parameter variations in both the high-performance and low-power NoC links. Concretely, this figure shows the cumulative distribution function of the maximum achievable link frequency for 32nm, 22nm, and 16nm. As shown in the figure, as technologies scale down, the impact of parameter variations is more noticeable. Note that lower slope means higher variability. The reason behind this is that random dopant fluctuations are dominating over the other components of variations for technologies below 32nm, and its contribution considerably increases as the size of transistors is reduced. Figure 3(a) shows that in the LP link, as a consequence of variations, maximum achievable wire frequency ranges from 1.5GHz to 3.1GHz (for a 16nm technology). In the case of HP links (Figure 3(b)), maximum wire achievable frequency varies between 2.5GHz and 4.1GHz (for a 16nm technology). The measured variation (σ/μ) for a 16nm technology of the maximum achievable frequency is 16.4% and 11.1%, for LP and HP links respectively. LP links suffer from higher variability as the required device size is lower.

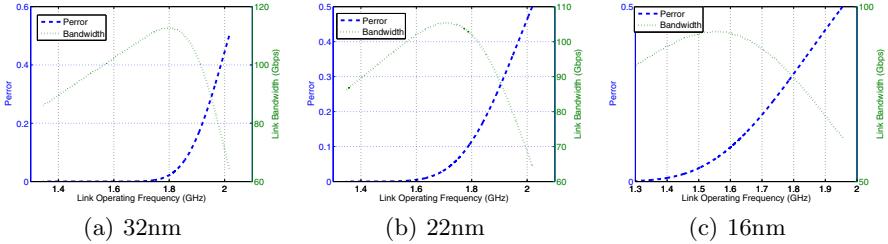


Fig. 4. Link error probability and maximum achievable bandwidth for Low-Power links

5.3 Performance Evaluation

In this section we compare the performance of the phit reduction (*PR*) approach with the traditional frequency reduction (*FR*) approach². Note that in order to keep all the link wires working, in a variable frequency NoC link design (*FR*), the frequency of the link must be lowered in order to switch at the frequency of the slowest wire. On the contrary, when our proposal (*PR*) is leveraged, higher frequencies can be used. For that purpose we have to tolerate the presence of faulty wires. In this sense, the variable phit-size NoC architecture allows the use of higher frequencies at the expense of discarding some slower wires. As a result the bandwidth of the link can be maximized by finding the optimal trade-off between link operating frequency and the number of faulty wires.

Figures 4 and 5 elaborate on the mentioned trade-off. These figures show the link error probability for 32nm, 22nm, and 16nm technologies, for both the HP and LP links, respectively. Additionally, Figures 4 and 5 also show link bandwidth when applying the *PR* mechanism at different frequencies. For a 32nm technology, the *LP* link achieves a maximum bandwidth of 112.7Gbps (Figure 4(a)), while the bandwidth of the *HP* is 186.6 Gbps (Figure 5(a)). On the contrary, for a 32nm technology the *FR* mechanism achieves is 86.0333 Gbps for the *LP* link, and 138.3 Gbps for the *HP* link. Notice that the nominal bandwidth, the bandwidth of a link where all wires work at nominal frequency, is 143.8 Gbps and 209.2Gbps, for the *LP* and *HP* designs, respectively.

Table 2 shows link bandwidth degradation caused by delay variation for the *LP* and *HP* links, respectively. Concretely, these tables compare the performance of *PR* and *FR* mechanisms in the presence of process variations. As shown in this table, using the *PR* mechanism, the impact of process variation in NoC links is reduced. Concretely, in the worst case scenario, 16nm technology and the *LP* link, using the *PR* approach the bandwidth is still 92.9 Gbps, a 65% of the nominal bandwidth. This bandwidth is an 11.1% higher than the one achieved by the *FR* approach.

Results confirm that a higher link bandwidth can be retrieved from links in the presence of variation using the *PR* approach. This mechanism allows to work at

² Note that as we are evaluating link performance the *PR* mechanism applies to both the *LPR* and the *GPR* approach.

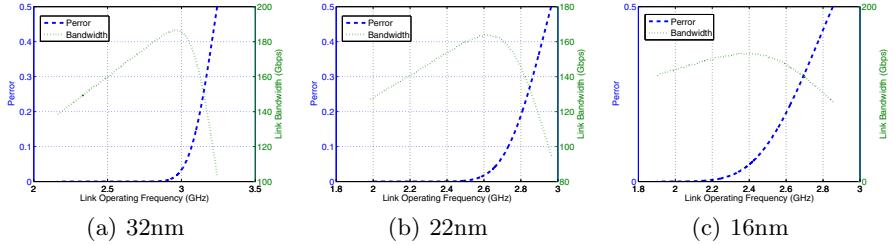


Fig. 5. Link error probability and maximum achievable bandwidth for High-Performance links

Table 2. Bandwidth and operating frequency for the *HP* and *LP* links using the *PR* and *FR* mechanisms

	32nm		22nm		16nm	
	<i>PR</i>	<i>FR</i>	<i>PR</i>	<i>FR</i>	<i>PR</i>	<i>FR</i>
HP Link	BW(Gbps)	186.5609	138.3107	163.6772	126.4592	146.1798
	Freq(GHz)	2.9618	2.1611	2.6213	1.9759	2.3864
LP Link	BW(Gbps)	112.6759	86.0333	105.3603	86.1243	92.8926
	Freq(GHz)	1.7997	1.3443	1.7112	1.3457	1.5526

higher frequencies by tolerating the presence of faulty wires. Furthermore, using the *PR* mechanism the presence of manufacturing defects is also tolerated, and, thus, the NoC yield is improved.

5.4 Area Results

In this section the area overhead of the proposed variable phit-size NoC architecture is computed. The area results shown in this section correspond to the implementation of the variable phit NoC architecture in a 45nm technology. The area overheads are due to the inclusion of the variable phit size flit injector and ejectors modules required, by both the *LPR* and *GPR* approaches, and by the required omega crossbars at every input and output external port of the NoC switches. Note that for an efficient omega network implementation a full custom design is required. Therefore, area costs of the omega network have been computed according to its transistor count which is $8 * \frac{N}{2} * \log_2(N)$. Table 3 shows the area overheads of the fault-tolerant architecture proposed for a 64-bit NoC implementation. In this table the area required by a 64-bit five port router architecture is also shown. The *LPR* mechanism requires an area that is 18% the area of a regular five port NoC switch. However, when using the *GPR* approach the area overhead of the proposal is reduced. Concretely, this latter approach increases switch area only by 9%.

As shown, the proposed variable phit-size NoC architecture requires non-negligible area resources. However, similar proposals that tolerate manufacturing

Table 3. Area overhead of the fault-tolerant architecture for a 64-bit flit NoC

	Area (μm^2)	Overhead
Switch	28356.72	-
LPR	5107	18%
GPR	2655.72	9.37%

faults are also costly in terms of area resources. In this sense, Table 4 shows a complexity comparison of our proposal and the use of spare wires [5]. This table shows the required cross-points to implement both approaches. Note that in this table the overhead of the variable phit-size NoC architecture in NICs is not computed. The results for the spare crossbar represent the cross-point requirements for achieving a link yield of 99% and 99.9% given a wire error probability of 0.01. On the contrary, the cross-points required for implementing the *PR* approach are fixed regardless the number of faulty wires, and, therefore, achieving a 100% link yield. Results of this table show how for wide links the use of an omega link crossbar requires less cross-points than the use of an spare crossbar [5]. Consequently, for a 128-bit wide link the spare crossbar requires $1.48 \times$ and $2.83 \times$ the number of cross-points of an omega link crossbar for achieving only 99% and 99.9% yield, respectively.

Table 4. Hardware cost comparison

Interconnect width	Spare(99%)	Spare(99.9%)	Omega Crossbar
32	288	708	640
64	548	2180	1536
128	5312	10128	3584

6 Conclusions

In this paper a new fault-tolerant NoC architecture is presented. The proposed architecture is based on variable phit-size injection and ejection of flits that allows the use of links presenting some faulty-wires. This variable phit-size NoC architecture is presented in two flavours: *LPR* and *GPR*. The *LPR* approach causes an important overhead of NoC area and is oriented for such manufacturing scenarios with high defect density levels and dominated by a strong systematic L_{eff} variation. On the contrary, the *GPR* reduces considerably the area overhead and is suitable for scenarios with low defect density and high random threshold variations.

The main advantages of the proposed variable phit size NoC architecture are the ability to tolerate both manufacturing defects and variation induced timing errors. On one hand, this approach achieves better performance than other variation tolerant link design approaches based on reducing the link frequency. Results shown in the evaluation section confirm that for 32nm, 22nm, and 16nm

technologies reducing the phit size retrieves more bandwidth from faulty links than reducing the frequency. Concretely, we achieve a 31% and 34.4% bandwidth improvement ratio over frequency reduction techniques for LP and HP links, respectively.

On the other hand, the *PR* mechanism also outperforms other proposals oriented to face the presence of faulty wires in NoC links, as is able to maximize yield at a reasonable area costs and regardless the amount and the origin of failures. Results shown that for 128-wide links our proposal requires less area than the use of spare wires, while the yield achieved is also superior.

References

1. Mejia, A., Fliech, J., Reinemo, S., Skeie, T., Duato, J.: Segment-Based Routing: An Efficient Fault-Tolerant Routing Algorithm for Meshes and Tori. In: 2006 International Parallel and Distributed Processing Symposium, IPDPS 2006 (2006)
2. Chen, G., Friedman, E.G.: Low-power repeaters driving RC and RLC interconnects with delay and bandwidth constraints. IEEE Transactions on Very Large Scale Integration Systems 14(2), 161–172 (2006)
3. Ernst, D., Kim, N.S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K., Mudge, T.: Razor: a low-power pipeline based on circuit-level timing speculation. In: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-36, pp. 7–18 (December 2003)
4. International Technology Roadmap for Semiconductors. ITRS 2007 Online Edition, <http://www.itrs.net/Links/2007ITRS/Home2007.htm>
5. Grecu, C., Ivanov, A., Saleh, R., Pande, P.P.: Noc interconnect yield improvement using crosspoint redundancy. In: 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT 2006, pp. 457–465 (October 2006)
6. Hernández, C., Silla, F., Duato, J.: A methodology for the characterization of process variation in noc links. In: Proceedings of the Design, Automation and Test in Europe Conference, pp. 685–690 (March 2010)
7. Hernández, C., Roca, A., Fliech, J., Silla, F., Duato, J.: Characterizing the impact of process variation on 45 nm noc-based cmps. Journal of Parallel and Distributed Computing 71(5), 651–663 (2011); Networks-on-Chip
8. Murali, S., Tamhankar, R., Angiolini, F., Pulling, A., Atienza, D., Benini, L., De Micheli, G.: Comparison of a timing-error tolerant scheme with a traditional retransmission mechanism for networks on chips. In: International Symposium on System-on-Chip, pp. 1–4 (November 2006)
9. Owens, J.D., Dally, W.J., Ho, R., Jayasimha, D.N., Keckler, S.W., Peh, L.: Research challenges for on-chip interconnection networks. IEEE Micro 27(5), 96–108 (2007)
10. Hassan, F., Cheng, B., Vanderbauwhede, W., Rodriguez, F.: Impact of device variability in the communication structures for future synchronous soc designs. In: International Symposium on System-on-Chip, SOC 2009, pp. 068–072 (October 2009)
11. Worm, F., Ienne, P., Thiran, P., De Micheli, G.: A robust self-calibrating transmission scheme for on-chip networks. IEEE Trans. Very Large Scale Integr. Syst. 13(1), 126–139 (2005)

Performance and Energy Efficiency Analysis of Data Reuse Transformation Methodology on Multicore Processor

Abdullah Al Hasib¹, Per Gunnar Kjeldsberg², and Lasse Natvig¹

¹ Department of Computer and Information Science

{abdullah.alhasib,lassie.natvig}@idi.ntnu.no

² Department of Electronics and Telecommunications

Norwegian University of Science and Technology, Trondheim, NO-7491, Norway

Abstract. Memory latency and energy efficiency are two key constraints to high performance computing systems. Data reuse transformations aim at reducing memory latency by exploiting temporal locality in data accesses. Simultaneously, modern multicore processors provide the opportunity of improving performance with reduced energy dissipation through parallelization. In this paper, we investigate to what extent data reuse transformations in combination with a parallel programming model in a multicore processor can meet the challenges of memory latency and energy efficiency constraints. As a test case, a “full-search motion estimation” kernel is run on the Intel® Core™ i7-2600 processor. Energy Delay Product (EDP) is used as a metric to compare energy efficiencies. Achieved results show that performance and energy efficiency can be improved by a factor of more than 6 and 15, respectively, by exploiting a data reuse transformation methodology and parallel programming model in a multicore system.

Keywords: Performance, energy efficiency, data reuse transformation methodology, parallel programming.

1 Introduction

The rapid growth of microprocessor performance for the last two decades has provided us the opportunity to solve increasingly advanced problems that require very large scale computations. However, memory latency has not been improved at a comparable rate and has become a major limiting factor for system performance. System performance is further impeded by battery capacity for handheld devices and by heat dissipation constraints for high performance processor designs [1]. Therefore, improvement of energy efficiency and memory latency has now become a major concern in contemporary computer architectures.

For data-dominated applications such as multimedia algorithms, Data Transfer and Storage Exploration (DTSE) offers a complete methodology for obtaining and evaluating a set of data reuse transformations in terms of memory energy [2].

The fundamental idea behind data reuse transformations is to move the data accesses from background memories to smaller and less energy intensive foreground memory blocks in a systematic way. This approach eventually results in significant energy savings.

In this paper, we present a technique that combines a data reuse transformation methodology with a parallel programming model to improve energy efficiency. We evaluate the performance and energy efficiency of our combined technique on a quad-core processor. We have used a “full-search motion estimation” algorithm as our test application.

This paper is organized as follows: Section 2 describes related work. Section 3 presents our methodology to improve energy efficiency and 4 illustrates our methodology using the *motion estimation* algorithm. Section 5 presents and discusses our results. Finally, we conclude the paper in Section 6.

2 Related Work

Research on data reuse transformation methodologies for multimedia applications has been actively performed for the last few decades and has led to numerous approaches to improve memory latency. Wuytack *et al.* [2] present a formalized methodology for data reuse exploration to reduce memory energy consumption by exploiting temporal locality of memory accesses using an optimized custom memory hierarchy. It is taken further and oriented towards pre-defined memory organizations in [3]. In [4,5], the authors evaluated the effect of data reuse decisions on power, performance and area in embedded processing systems. The effect of data reuse transformations on a general purpose processor has been explored in [6]. In [7], the authors presented the effect of data reuse transformations on multimedia applications on application specific processors. The research described so far emphasizes on single-core systems and relies on simulation based modeling when energy efficiency is estimated.

In [8], Kalva *et al.* presented the effect of parallel programming on multimedia applications but it lacks energy efficiency analysis. In [9], Chen *et al.* presented different optimization opportunities of the Fast Fourier Transform (FFT) to achieve a high performance implementation on IBM Cyclops-64 chip architecture. In [10], Zhang *et al.* presented an inter-core data reuse technique to exploit all the available cores to boost overall application performance. These earlier studies on parallel architectures emphasize performance rather than energy efficiency. In contrast, we have performed energy efficiency analysis on a state-of-the-art multicore processor with four cores and have used Model Specific Registers of the processor to accurately measure the consumed energy.

The previous work closest to ours is done by Marchal *et al.* [11]. It presents an approach for integrated task-scheduling and data-assignment for reducing SDRAM costs for multi-threaded applications. It does not couple the data reuse analysis with a parallel programming model the way we do here, however.

3 Energy Efficient Methodology for Multicore Processor

In this paper, we present an approach that combines the concepts from a *data reuse transformation* methodology with a *parallel programming* model to get better performance and energy efficiency.

Data Reuse Transformation: The fundamental concept of a data reuse transformation is to optimize an application and/or introduce a custom memory hierarchy to exploit the temporal locality of data accesses [2]. The memory hierarchy consists of layers of gradually smaller memories. The application code is optimized so that data that is accessed multiple times, i.e., has a high reuse factor, is copied from larger to smaller memories closer to the data path. Unless the memory hierarchy is fixed, the size and interconnect of each layer can also be optimized. For data-intensive applications, this approach gives significant energy savings since smaller memories consume less energy per access.

Parallel Programming: Multicore processors can achieve higher performance with lower energy consumption compared to a uniprocessor system. It is, however, a challenging job to develop efficient parallel applications that exploit the advantages of hardware parallelism. Different parallel programming models have been developed that can speed up applications when multiple threads or multiple processes are used [12]. At this level, parallel programs can be written using multi threaded programming and using explicit threading supported by the operating system or using programming frameworks such as OpenMP [13].

In our approach, initially we have applied different possible data reuse transformations described in [2] to optimize energy efficiency for a given algorithm. The first step identifies data sets that are reused multiple times within a short period of time, i.e., *copy candidates*. For each of the identified data sets, a copy to a smaller memory can be introduced so that data is accessed using less energy. Based on a cost trade off with extra copying of data and chip area overhead, a hierarchical memory organization is generated and an optimized set of *copy candidates* are utilized. After data reuse optimization, we develop a parallel algorithm based on the optimized solution.

4 Demonstrator Application: Motion Estimation Kernel

We have used a “full-search motion estimation” algorithm to evaluate the performance and energy efficiency of our combined approach.

4.1 Sequential Unoptimized Motion Estimation Algorithm

Motion Estimation (ME) is a core part of different video compression algorithms. Block-based ME algorithms involve finding the candidate block within a specified search area in a *reference frame* that is most similar to the current block in the *current frame*. A “full-search motion estimation” algorithm performs an exhaustive search over the entire search region to find the optimal solution.

Algorithm 1. Sequential Unoptimized Motion Estimation Algorithm [14]

```

1: for  $g=0$ ;  $g < H/n$ ;  $g++$  do
2:   for  $h=0$ ;  $h < W/n$ ;  $h++$  do
3:      $\Delta_{opt}[g][h] = +\infty$ 
4:     for  $i=-m$ ;  $i < m$ ;  $i++$  do
5:       for  $j=-m$ ;  $j < m$ ;  $j++$  do
6:          $\Delta = 0$ 
7:         for  $k=0$ ;  $k < n$ ;  $k++$  do
8:           for  $l=0$ ;  $l < n$ ;  $l++$  do
9:              $\Delta += \text{abs}(\text{Cur}[g \times n+k][h \times n+l] - \text{Ref}[g \times n+i+k][h \times n+j+l])$ 
10:            end for
11:          end for
12:           $\Delta_{opt}[g][h] = \min(\Delta, \Delta_{opt}[g][h])$ 
13:        end for
14:      end for
15:    end for
16:  end for

```

This process is computationally intensive and costs about 80% of the encoding time [8]. Therefore, we have chosen it as a test application in our experiment.

Full-search motion estimation is illustrated in Algorithm 1. The implementation of the ME algorithm consists of a number of nested loops. The basic operation at the innermost loop consists of an accumulation of pixel differences, while the basic operation two levels higher in the loop hierarchy consists of the selection of the new minimum. This algorithm is a sequential implementation without exploiting any data reuse transformation technique and referred as *sequential unoptimized* solution in this paper. For our experiment, we have used parameters of the QCIF format ($W=176$, $H=144$, $m=n=8$) [14].

4.2 ME Optimization Using Data Reuse Transformations

We have followed a systematic approach presented in [2] to transform the Basic ME Algorithm into an optimized solution that maps selected copies of data on a memory hierarchy to exploit temporal locality. Fig. 1 presents different possible transformations for the ME algorithm.

Each branch in the *copy candidate* tree corresponds to a potential memory hierarchy for different data-reuse transformations. Dashed lines in the figure indicate levels of the hierarchy. Each rectangle in the hierarchy corresponds to a *copy candidate*, i.e., a block of data that can benefit from being accessed multiple times from the given hierarchy level. Each *copy candidate* is annotated with its size. The highlighted path in the figure indicates a *3 layer memory* hierarchy for data reuse transformations on the *reference frame*. The hierarchy is comprised of a $H \times W$ block for the full frame, a $(2m+n-1) \times (2m+n-1)$ block and a $(2m+n-1) \times n$ block for smaller *copy candidates*. In addition, a *2 layer memory* hierarchy for the *current frame* with a $H \times W$ frame memory and a $n \times n$ *copy candidate* is also introduced.

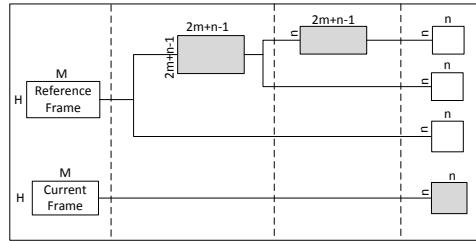


Fig. 1. Copy candidate tree for data reuse decision for Motion Estimation Algorithm. The process of constructing such copy candidate tree is explained in [2].

To evaluate the performance and energy efficiency of the different data reuse transformations presented in Fig. 1, the basic ME algorithm has been modified into different versions to exploit different possible transformations. Achieved performance and energy efficiency of all transformed algorithms are then measured and compared. The transformation that provides the best energy efficiency is converted to a parallel program. Algorithm 2 depicts an example of the transformed ME algorithm with two layers. The transformed algorithm introduces a smaller memory block (*Buffer*) to which the *copy candidate* is copied.

4.3 Parallel Optimized Motion Estimation Algorithm

The Motion Estimation algorithm also exhibits important properties of data parallelism. In QCIF format, a video frame is comprised of a fixed number of macro blocks (8×8 non-overlapping blocks). Prediction for a given block is determined by finding a block in a given search range of the *reference frame* that is closest to the current block. For each macro block (MB), this estimation can be done in parallel. Algorithm 2 represents our parallel ME algorithm. We have made our *2 layer* ME algorithm parallel by adding the *#pragma omp parallel for* directive of the OpenMP programming model [13] at the outermost *for* loop. This directive will instruct the compiler to distribute the work done in the for-loop immediately following the directive among all processors (cores) of the system. Variables *Ref*, *Cur* and Δ_{opt} are shared among the threads whereas (*h*, *Buffer*) are private to each thread. Note that threads should be properly synchronized while computing the minimum Δ_{opt} . Therefore a *#pragma omp critical* directive is used to ensure that Δ_{opt} is accessed by a single thread at a time. We have also set the *GOMP_CPU_AFFINITY* environment variable to bind each thread, i.e., each instance of the *for-loop*, to a specific core.

4.4 System Architecture and Energy Measurement

System Architecture: In our experiment, we have used the Intel® Core™ i7-2600 processor which consists of four physical cores. It supports Hyper-Threading allowing it to simultaneously process up to 8 threads, i.e., 2 threads per core.

Algorithm 2. Parallel Optimized Motion Estimation Algorithm

```

1: #pragma omp parallel for shared(Ref, Cur,  $\Delta_{opt}$ ) private(h, Buffer)
2: for  $g=0; g < H/n; g++$  do
3:   for  $h=0; h < W/n; h++$  do
4:     for  $k=0; k < 2m+n-1; k++$  do
5:       for  $l=0; l < 2m+n-1; l++$  do
6:          $Buffer[k][l] = Ref[g \times n-m+k][h \times n-m+l]$ 
7:       end for
8:     end for
9:      $\Delta_{opt}[g][h] = +\infty$ 
10:    for  $i=0; i < 2m-1; i++$  do
11:      for  $j=0; j < 2m-1; j++$  do
12:         $\Delta = 0$ 
13:        for  $k=0; k < n; k++$  do
14:          for  $l=0; l < n; l++$  do
15:             $\Delta += abs(Cur[g \times n+k][h \times n+l] - Buffer[i+k][j+l])$ 
16:          end for
17:        end for
18:        #pragma omp critical
19:         $\Delta_{opt}[g][h] = min(\Delta, \Delta_{opt}[g][h])$ 
20:      end for
21:    end for
22:  end for
23: end for

```

The memory hierarchy consists of a 32 KB Level-1 cache, a 256 KB Level-2 cache and an 8192 KB Level-3 cache. Level-1 and Level-2 caches are private to each core while the Level-3 cache is shared among the cores. Note that this is a memory hierarchy with a fixed number of levels and sizes, typical for a standard processor. This is different from the assumption in [2], where an application specific memory hierarchy is assumed. The base clock speed of the processor is 3.4 GHz, but it can go as high as 3.8 GHz when Turbo Boost is enabled [15].

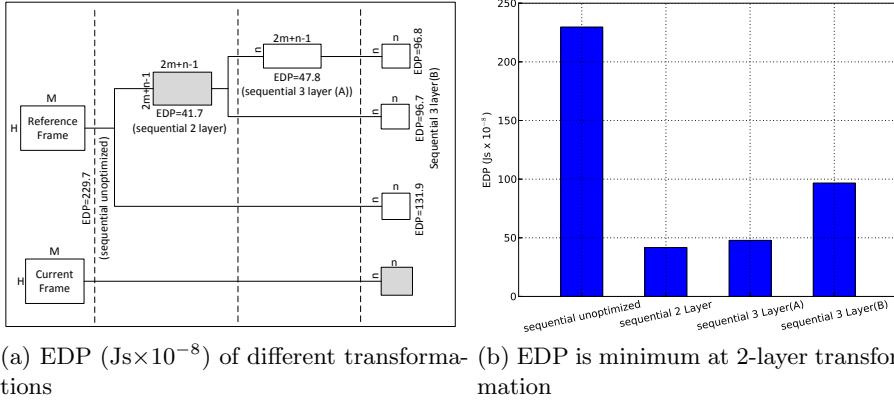
Energy Measurement Policy: We read the non-architectural Model Specific Registers of the processor to estimate on-chip energy consumption [15]. The *MSR_PP0_ENERGY_STATUS* register gives us aggregate energy consumed by the cores as well as caches. We read this register at a fixed core frequency (3.4 GHz) and process the raw data to compute energy efficiency.

Energy Efficiency Metric: We report energy efficiency in terms of the Energy-Delay-Product (EDP) metric [16]. Measured units for Energy and Delay are Joule(J) and second(s) respectively. Therefore, the unit of the EDP metric is $J.s$. Generally, the lower the EDP, the better the energy efficiency is.

OS and Compiler Parameters: We execute our experiment on OpenSuse 11.4 (x86 64) running Linux kernel 2.6.37.6. The parallel application is compiled using the *gcc* compiler with *-fopenmp* flag and optimized with *-O3* flag.

5 Results and Discussion

Energy Efficiency Evaluation of Sequential ME Algorithm: Different data reuse transformations of the sequential ME algorithm and their corresponding energy efficiencies are presented in Fig. 2.



(a) EDP ($\text{Js} \times 10^{-8}$) of different transformations (b) EDP is minimum at 2-layer transformation

Fig. 2. Data reuse transformations and their energy efficiencies measured in EDP

Fig. 2a shows that energy efficiency is improved significantly due to data reuse transformation techniques despite of the fact that such transformations introduce both area and computational overheads. For instance, the *sequential 2 layer* transformation introduces a $(2m+n-1) \times (2m+n-1)$ block buffer for the *reference frame* and a $(n \times n)$ block buffer for the *current frame* and these additional buffers cost 2372 Bytes of area overhead. The computational and energy overhead to copy the *copy candidates* into the buffer are 0.31 microsecond and 6.67 millijoule, respectively. Therefore, in terms of EDP, the overhead is approximately 0.207×10^{-8} Js for each *new frame*. Despite these overheads, we have observed that achieved EDP for the complete handling of one *new frame* is 229.7×10^{-8} Js for the *sequential unoptimized* ME Algorithm whereas the EDP of the *sequential 2 layer* transformation is 41.7×10^{-8} Js. This improvement is attributed to the use of smaller buffers since a block of $(2m+n-1) \times (2m+n-1)$ integer-numbers corresponds to 2116 ($23 \times 23 \times 4$) bytes which is less than the Level-1 cache size in our system. As a result, the buffer can be brought into the Level-1 cache during the computation which significantly reduces the cost of expensive memory accesses and improves performance as well as energy efficiency.

An important observation from Fig. 2b is that the efficiency is at a peak with a *2 layer memory* hierarchy and it degrades with the introduction of any additional layers of smaller memory blocks. Two factors that contribute to this result are: (i) Additional memory layers also introduce additional area and computational overheads (ii) Smaller data blocks are copied into relatively larger cache-blocks

due to the fixed-sized caches, which ultimately negate the advantage of using additional memory layers.

Energy Efficiency Evaluation of Parallel ME Algorithm: To maximize the energy efficiency, we have converted the optimized ME algorithm that uses a *2 layer memory hierarchy* into a parallel one by using the OpenMP programming model and executed it on our system with a varying number of threads. Fig. 3 presents the obtained result.

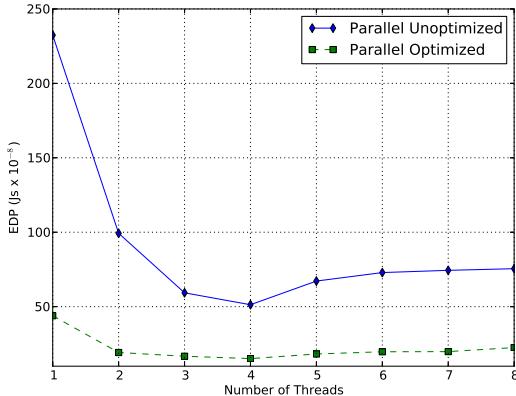


Fig. 3. Improved energy efficiency using optimized parallel ME algorithm

Fig. 3 implies that parallel programming improves energy efficiency of both optimized (that exploits data reuse transformation methodology) and unoptimized solutions (not using data reuse transformations). We can see that EDP values drop rapidly with increasing number of threads and reach their minimum when 4 threads are used. Since the Intel® Core™ i7-2600 processor consists of 4 physical cores which are shared among the threads in Hyper-Threading mode, cache pollution causes the *parallel unoptimized* solution to increase the EDP values with the increasing number of threads. In contrast to the unoptimized solution, the optimized solution exhibits better cache behavior due to the use of smaller memory blocks. Hence, EDP remains almost constant during the Hyper-Threading mode.

Table 1 presents a summary of our results which reveal that data reuse transformations significantly improve energy efficiency and that the *parallel optimized* solution is the most energy efficient transformation for ME algorithm. Normalized EDP values (with respect to *optimized parallel solution*) in the Table indicate that, *sequential optimized* and *sequential unoptimized* solutions are energy in-efficient by a factor of 2.7 and 15.1, respectively. The execution time for performing ME on one complete *new frame* is improved with a factor of 6.5 going from *sequential unoptimized* to *parallel optimized*.

Table 1. Results of different data reuse transformations

Version	Execution Time Second $\times 10^{-6}$	Energy Joule $\times 10^{-3}$	Energy Efficiency (EDP) Js $\times 10^{-8}$	Normalized EDP
Sequential Unoptimized	10.9	210.7	229.7	15.1
Sequential 2 Layer	4.3	97.0	41.7	2.7
Sequential 3 Layer	4.6	104.1	47.8	3.1
Parallel Unoptimized	3.2	161.4	51.6	3.4
Parallel Optimized	1.7	89.7	15.2	1.0

In contrast to our results, in which we have obtained the best energy efficiency by using a *2 layer memory* hierarchy, Wuytack *et al.* in [2] have shown that a *3 layer memory* hierarchy is the most energy efficient scheme for the ME algorithm. However, our experiments differ from their as follows: First, we have experimented on a processor with a memory hierarchy of fixed sized cache-blocks. The *copy candidates* are hence mapped to a portion of these fixed-size system caches, and consequently our measurements consider the energy consumed by both used and idle cache lines. Wuytack *et al.* did their experiment in a simulation environment that created a hierarchy of memory blocks that perfectly fit the data blocks. Therefore, extra energy consumption due to unused cache area is avoided. To avoid extra energy consumption in our experiment, we would need to have an execution platform using a concept like *drowsy cache* [17] that powers down unused parts of the cache. This would give more comparable results between the two methods. It is not available in the CoreTM i7 processor, however. Second, we have measured energy efficiency of the complete program rather than a part of the program that deals with data transfer. Third, we have measured on-chip memory and core energy consumption rather than considering only memory energy consumption. Fourth, their simulation environment assumes that data can be directly copied from a low-level hierarchy to a high-level hierarchy bypassing any intermediate layer. This is not possible in our system.

6 Conclusion

In this paper, we have investigated performance and energy efficiency effects of applying data-reuse transformations on a multicore processor running a motion estimation algorithm. We have shown that for a sequential *Motion Estimation* kernel, energy efficiency can be improved up to 5.5 times by using appropriate data-reuse transformation techniques, which can be further extended to 15.1 times by incorporating the OpenMP parallel programming model. We have also shown that Hyper-Threading degrades both performance and energy efficiency of the unoptimized solution. This gives clear indications that a data reuse transformation methodology in combination with a parallel programming model can significantly save energy as well as improve performance of this type of applications running on multicore processors.

References

1. Albers, S.: Energy-Efficient Algorithms. *Communications of the ACM* 53(5), 86–96 (2011)
2. Wuytack, S., Diguet, J.P., Catthoor, F., et al.: Formalized Methodology for Data Reuse Exploration for Low-Power Hierarchical Memory Mappings. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 6(4), 529–537 (1998)
3. Catthoor, F., Danckaert, K., Kulkarni, K., et al.: Data Access and Storage Management for Embedded Programmable Processors. Kluwer Academic Publishers, Dordrecht (2002)
4. Catthoor, F., Wuytack, S., de Greef, G., et al.: Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design. Kluwer Academic Publishers, Norwell (1998)
5. Zervas, N.D., Masselos, K., Goutis, C.E.: Data-Reuse Exploration for Low-Power Realization of Multimedia Applications on Embedded Cores. In: Proc. 9th International Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS 1999, pp. 71–80 (1999)
6. Chatzigeorgiou, A., Chatzigeorgiou, E., Kougiou, S., et al.: Evaluating the Effect of Data-Reuse Transformations on Processor Power Consumption (2001)
7. Vassiliadis, N., Chormoviti, A., Kavvadias, N., et al.: The Effect of Data-Reuse Transformations on Multimedia Applications for Application Specific Processors. In: Proc. Intelligent Data Acquisition and Advanced Computing Systems Technology and Applications, IDAACS 2005, pp. 179–182 (September 2005)
8. Kalva, H., Colic, A., Garcia, A., et al.: Parallel Programming for Multimedia Applications. *Multimedia Tools and Applications* 51(2), 801–818 (2011)
9. Chen, L., Hu, Z., Lin, J., et al.: Optimizing the Fast Fourier Transform on a Multi-core Architectures. In: Proc. Parallel and Distributed Processing Symposium, IPDPS 2007, pp. 1–8 (March 2007)
10. Zhang, Y., Kandemir, M., Yemliha, T.: Studying Inter-core Data Reuse in Multicores. In: Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2011, pp. 25–36 (2011)
11. Marchal, P., Catthoor, F., Bruni, D., et al.: Integrated Task Scheduling and Data Assignment for SDRAMs in Dynamic Applications. *IEEE Design & Test of Computers* 21(5), 378–387 (2004)
12. Podobas, A., Brorsson, M., Faxén, K.F.: A Comparison of some recent Task-based Parallel Programming Models. In: Proc. 3rd Workshop on Programmability Issues for Multi-Core Computers, Pisa, Italy (January 2010)
13. OpenMP Architecture Review Board: OpenMP Application Program Interface (July 2011), <http://www.openmp.org/mp-documents/OpenMP3.1.pdf>
14. Komarek, T., Pirsch, P.: Array Architectures for Block Matching Algorithms. *IEEE Transactions on Circuits and Systems* 36(10), 1301–1308 (1989)
15. Intel: Intel 64 and IA-32 Architectures Software Developer’s Manual (2011)
16. Rivoire, S., Shah, M.A., Ranganathan, P., et al.: Models and Metrics to Enable Energy-Efficiency Optimizations. *Computer* 40(12), 39–48 (2007)
17. Flautner, K., Kim, N.S., Martin, S., et al.: Drowsy Caches: Simple Techniques for Reducing Leakage Power. In: Proc. 29th Annual International Symposium on Computer Architecture, ISCA 2002, Washington, DC, USA, pp. 148–157 (2002)

Effects of Process Variation on the Access Time in SRAM Cells

Vicent Lorente and Julio Sahuquillo

Department of Computer Engineering (DISCA)
Universitat Politècnica de València, Spain
`{vlorente, jsahuqui}@disca.upv.es`

Abstract. As technology advances continue reducing transistor features, microscopic variations in number and location of dopant atoms in the channel region induce increasing electrical deviations in device parameters such as the threshold voltage. Deviations refer to mismatches with respect to device parameters at design time. These deviations are specially important in SRAM cells whose transistors are constructed with minimum geometry to fulfill area constraints, since they can cause some cells to fail.

In this paper, we study the impact of threshold voltage variations in the stability of the cell for a 16nm technology node. The failure probability has been studied for the four types of SRAM failures: write, access, read, and hold. We found that, under the assumed experimental conditions, the two former types of failures can be reduced by increasing the wordline pulse width of the cell. Experimental results show that access failures can be reduced up to 43.9% and write failures around 23.4% by enlarging the wordline pulse by 5 times the nominal width.

1 Introduction

As technology node continues to shrink, dealing with manufacturing imperfections is a major design concern since they affect the manufacturing yield, the energy consumption, and the performance of current and incoming processors.

Because of process variation, the manufacturing process provides transistors with different features (e.g. threshold voltage, channel length, or channel width) so that not all the transistors in a chip are able to properly work with the same voltage and frequency conditions. Due to this fact, manufacturers opt for relaxing the conditions to introduce in the market those chips having a noticeable amount of transistors not able to properly work under the target design goal. For instance, some processors are sold at a lower price when their speed is lowered below the target one to avoid process variation errors.

Cache structures occupy a large area of the microprocessor die. To reduce this area, SRAM cells of caches are designed with the minimum transistor size. This design condition makes these cells particularly sensitive to voltage scaling.

As a consequence, dealing with process variation in cache memories is a critical design issue. Failures due to the manufacturing process in caches mainly rise in

destructive reads, unsuccessful writes, an increase in the access time, and content destruction in stand-by mode; known as read, write, access time and hold failures, respectively.

The number of failures due to process variation is determined by the processor working conditions (power supply and frequency). In other words, most errors usually appear in a subset of the working conditions range. For instance, it may happen that an error reading a memory cell appears when the processor works at a given frequency, but that error might not appear when working at a slower frequency (i.e. access time failure). On the other hand, a destructive read could be avoided by increasing the voltage supply.

Understanding why errors appear and which conditions should be done to avoid them (or most of them) is important for microprocessor architects in order to take the proper architectural design choices to achieve the best tradeoff between performance and power.

Among the different transistor features, the most significant source of random intra-die variation is the threshold voltage. In this paper we characterize the four mentioned failures, varying the power supply for a wide range of voltage values.

In this paper we focus on time dependent types of failures, which can be reduced by enlarging read and write operations (i.e. longer WL-pulse width). At first sight, it may seem that longer operations may increase execution time, but indeed, it may not. The cell probability of failure affects directly to the cache capacity. That is, the higher the probability of failure, the lower the effective cache capacity. As the effective cache capacity reduces, more cache misses are produced and therefore, the execution time increases. However, if we are able to reduce the probability of failure by reducing frequency, the execution time could not be affected because although a longer cache access time is used, the effective cache capacity is also larger. Therefore there is a tradeoff among cache access time, probability of failure (effective cache capacity) and energy. This paper is aimed at providing some preliminary results addressing these issues in order to help computer architects to devise the best design choice according to storage requirements of the running workloads.

This paper provides an study of the relation between WL pulse widths and SRAM time dependent failures. The remainder of this paper is organized as follows. Section 2 introduces the different types of SRAM cell failures. Section 3 characterizes SRAM cell failures at different power supply voltages, taking into account V_{th} variations. Section 4 presents the related work. Finally, Section 5 concludes the paper.

2 Background on SRAM Failures

Manufacturing process produces variations in the transistor parameters mainly due to physical factors caused by processing and masking imperfections [1]. Variations affect the channel length, channel width, oxide thickness, threshold voltage, line-edge roughness, and random dopant fluctuations, and are typically classified in *inter-die* and *intra-die* variations.

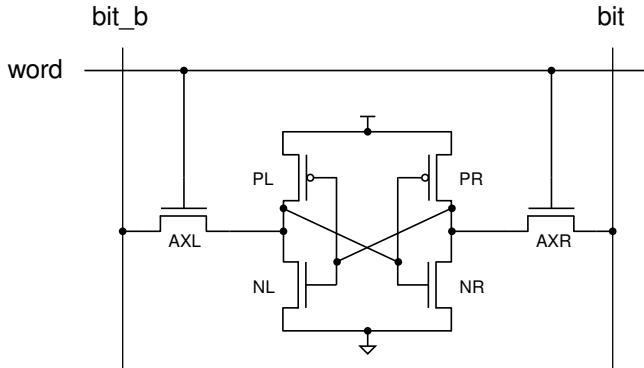


Fig. 1. 6T SRAM cell details

Inter-die variations affect all the transistors of a given die in the same way (e.g. threshold voltage of all the transistors either increase or reduce). As opposite, *intra-die* variations may affect transistors in the same chip in a different way (e.g. the V_{th} of some transistors can increase with respect to the nominal one whereas some others can have a V_{th} lower than the nominal one). In turn, *intra-die* variations can be either systematic or random. *Systematic* are variations depending on the variations of neighboring transistors while *random* variations are independent of the neighboring transistors.

Variations in different device parameters result in large spread in transistor threshold voltage [2]. Among them, the random placement of dopants causes threshold voltage mismatches among transistors that are spatially close to each other. Because of the small geometry of the SRAM cell, the main source of the device mismatch is the intrinsic fluctuation of the V_{th} of different transistors due to random dopant fluctuations [3–5], that is, random intra-die variations. These device parameters mismatches severely affect SRAM cells in sub-50nm technologies [6].

Each SRAM cell contains two pairs of transistors forming a logical not. Figure 1 shows these two pairs, one formed by PR-NR and the other formed by PL-NL. Any mismatch between devices of a pair degrades the stability of the cell and results in a cell failure when working at voltages lower than the design one. These mismatches between the variations of close transistors caused by intra-die variations can result in the failure of the cell in four different ways: hold failure, read failure, write failure, and access failure. Below we discuss them.

2.1 Hold Failure

Each of the mentioned transistor pairs is referred to as a node. One of them contains a “1” and the other a “0”. The voltage of the node storing “1” is the same as the power supply of the cell. Most current microprocessors implement a low power mode which highly reduces the power supply to save energy.

When working at this mode, if the voltage of the node storing “1” is reduced below the trip-point¹ of the node storing “0” then a flip occurs, so loosing the stored value and producing a hold failure.

2.2 Read Failure

Before the read is performed, both bitlines (i.e. *bit* and *bit_b*) must be precharged to Vdd. When the wordline is activated, the pass-transistors communicate both bitlines with the nodes of the cell. Then, the node storing “0” discharges the associated bitline while the node storing “1” remains to Vdd. The voltage increases for a while in the node storing “0” to a positive value due to the voltage divider action. When this increase is greater than the trip-point of the node storing “1”, a flip is produced, which is known as a read failure, since it occurs when the cell is being read.

2.3 Write Failure

In a write operation, the bitline is precharged to “0” or “1” according to the value to be written. A write failure is produced when a “0” cannot be written in the cell. When the wordline is activated, the pass-transistor communicates the node storing “1” (Vdd) with the bitline (0V). To be a successful write operation, the node storing “1” must reduce the voltage below the trip-point of the node storing “0” while the wordline is active. Due to process variation, this decrease may be too slow. In other words, causing the time the wordline is active is not longer enough to decrease the voltage below the trip-point.

2.4 Access Failure

The cell access time is defined as the time required to produce the necessary voltage difference to excite the sense amplifier in a read operation. This voltage difference is typically by 10% the Vdd and must be reached while the wordline is active. To perform a read, both bitlines are precharged to Vdd and the bitline of the node storing “0” is discharged to 0V. The time needed to discharge that bitline depends on the pass transistor and the NMOS features. Due to process variation, the mismatch in these transistors can affect to the discharging speed. If this speed is too slow, the difference needed to excite the sense amplifier is not achieved.

3 Experimental Evaluation

3.1 Methodology

Failure probabilities have been estimated simulating an SRAM cell with the HSPICE circuit level simulator. Experiments assumed SRAM transistors based

¹ The trip-point is the voltage necessary at the input of the node to change the output.

on 16nm nodes with high-performance profile from the Predictive Technology Model (PTM) [7]. Simulations used the BSIM4 MOSFET model that addresses the MOSFET physical effects into sub-100nm regime.

Transistor sizes have been chosen according to [8] to ensure read and writeability as well as to provide a good layout. Regarding area, device parameters (channel width W and channel length L) relationships (W/L) for the different types of transistors in the cell have been modeled as $6/2 \lambda$ for the access transistors², $4/2 \lambda$ for the pull-up PMOS transistors, and $8/2 \lambda$ for the pull-down NMOS transistors. To simulate read and write operations we assumed 90ps word-line pulses.

As stated in Section 2, the different random intra-die variations can be summarized in V_{th} fluctuations. These fluctuations have been modeled for each transistor (NMOS and PMOS) of the cell as an independent Gaussian random variable with μ and σ_{VTO} equal to 0 and 37.3% , respectively, and assuming a maximum V_{th} deviation equal to 112% [9]. For the sake of accuracy, we gathered 40000 samples of cells generated using the Monte Carlo simulation.

3.2 Error Failure Characterization

Power consumption strongly depends on the power supply. More precisely, dynamic and static power grow quadratically and linearly with the power supply, respectively. Thus many research has focused on reducing the power supply to reduce energy. Nevertheless, reducing power supply negatively impacts on the number of errors so hurting the performance. Therefore, process variation introduces a tradeoff between performance and power.

This section characterizes the behavior of the SRAM cells, quantified in probability of failure, varying the power supply and taking into account V_{th} variations.

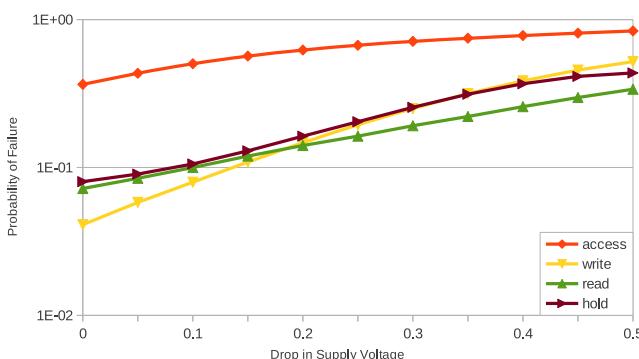


Fig. 2. Probability of failure reducing the power supply from 0.7V

² λ is defined as half the feature size (for 16nm nodes, $\lambda=8\text{nm}$).

Figure 2 shows the probability of failure for the four types of SRAM errors described above as the voltage supply drops. The nominal power supply has been set to 0.7V and drops are drawn in steps of 50mV; that is, the X axis shows the power supply ranging from 0.7V to 0.2V.

As observed, the access failure is the predominant one. In fact, this operation already shows a noticeable amount of failures at nominal voltage. Read, write and hold failures have a probability much smaller. This means that the major performance benefits can be achieved by attacking and reducing the access failures.

On the other hand, access failures, as well as write failures, are affected by the WL pulse time. That is, these errors can be reduced by using a larger WL pulse.

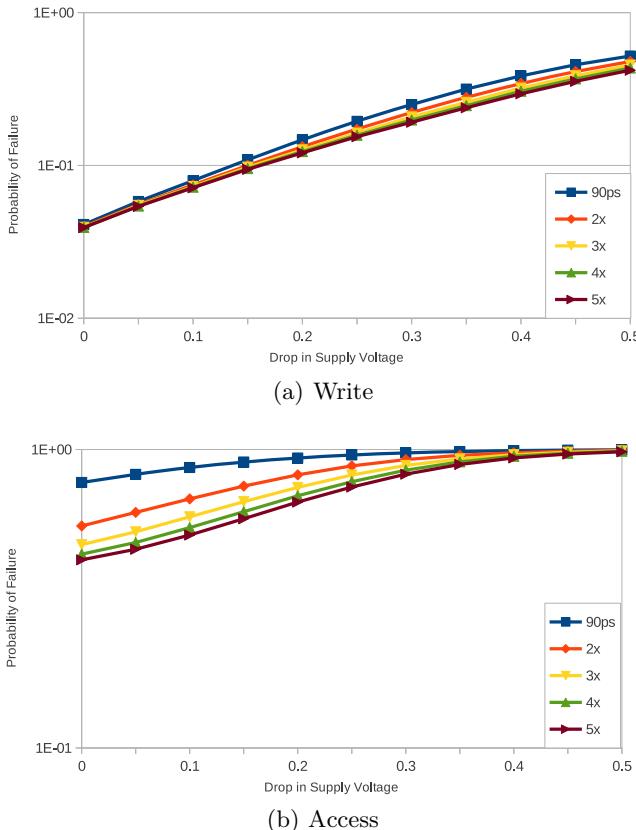


Fig. 3. Probability of write and access failures varying the WL pulse

3.3 Impact of the Access Time on Write and Access Failures

To estimate the pulse width the cell was tested using transistors with no variations and a power supply of 0.7V. The adequate pulse width is that that meets the specifications, that is, a WL pulse that gets a voltage difference between bitlines higher than 10% of Vdd. Attending to the results, we used a 90ps as the nominal pulse width.

Nevertheless, as discussed above, write and access failures are produced because, due to transistor variations, there is not enough time to carry out the operation. This section analyzes how longer pulses can help reducing write and access failures. To this end, we enlarged the WL pulse in a 2-, 3-, 4-, and 5x factor.

Figure 3 shows the probability of failure for access and write operations varying the WL pulse. Results are shown for WL pulses as large as 2-, 3-, 4- and 5x the original WL pulse length. As observed, differences among pulse lengths curves rise with low voltage drops in access operations (left side of Figure 3(b)) and with high voltage drops in write operations (right side of Figure 3(a)). Regarding access failures, in it can be observed (Figure 3(b)) that curves for the different pulse lengths begin to converge in a 0.4V ranging power-supply drop. That is, from such a drop no improvement can be done. On the other hand, in Figure 3(a), improvements appear with power supply drops from 0.15V to 0.5V.

Computer architects need insights on access time and probability of failures since depending on the workload behavior it could be a better choice a larger access time but with less failures than a shorter access time with a larger amount of failures or vice versa. These design issues are addressed next. To this end, Figures 4 and 5 present a zoom of the most interesting parts of Figure 3. Results on these figures can be analyzed in two main ways. By drawing a vertical line crossing the different curves, it can be estimated how much the pulse length can improve the probability of failure. On the other hand, by drawing an horizontal

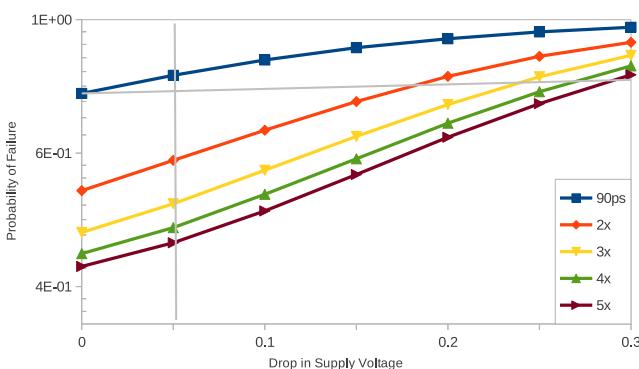


Fig. 4. Probability of access failure (zoom)

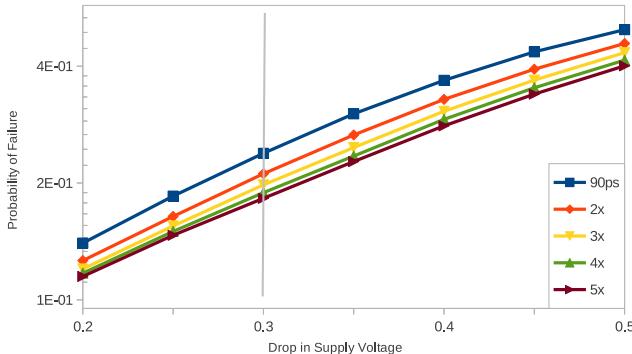


Fig. 5. Probability of write failure (zoom)

line, the curve traversing the crossing point identifies the pulse width required to work at a given voltage for a given probability of failure.

For instance, the vertical line drawn in the Figure 4 indicates that, for 0.65V, the probability of failure can be reduced around 25.5% by doubling pulse length, and as much as 43.9% by enlarging 5 times the nominal pulse.

Although the probability of failure increases with the voltage drop, the plotted horizontal line shows WL pulse length that would be required in order to keep the same probability of failure. For instance, for a 0.2V voltage drop, the required pulse to keep the same probability of failure as the original voltage should be twice as large the original one.

Regarding write operations Figure 5 shows that enlarging the WL pulse provides a rather more limited improvement than in access operations.

The vertical line shows that the probability of failure can be reduced about 11.4% reducing frequency 0.5X, and up to 23.4% by reducing it 5 times.

4 Related Work

Memory arrays blocks can be identified as unreliable at the post-fabrication process. Most of current approaches can be classified in two main types: those that correct unreliable blocks and those that avoid the use of it.

Regarding the former approaches type, Kim et al. [10] use error correcting codes (ECC) and redundancy to improve memory cell failures in SRAM caches, considering both systematic and random intra-die variations. This study takes into account three device parameters; the channel width, the channel length, and the threshold voltage. In contrast, we study how to reduce the number of two types of SRAM failures (access and write) in some cells increasing the WL-pulse width. In [11] Somnath et al. classify memory blocks in three main groups and apply different error correcting codes (ECC) to restore blocks according to the group they belong to. They also simulate effects of both inter and intra die variations.

Regarding the latter approaches type, Wilkerson et al. [12] propose two architectural techniques that reduce the effective cache storage capacity by 50%, based on probability of failure of a 65nm SRAM cell and taking into account V_{th} variations.

In [8] Mukhopadhyay et al. model failure probability of an SRAM cell in a 50nm technology node. For this purpose, random intra-die variations are taken into account resulting in V_{th} fluctuations. An exhaustive analysis of the different types of SRAM failures and their correlations is also provided. Their objective is to size the transistors of the SRAM cell at design time to minimize the failure probability of a memory chip considering area and leakage constraints. In contrast, our work keeps the same transistor size and varies the required time to access the cell to minimize the error failures caused by the reduction of power supply.

A model of timing errors due to parameter variation is proposed in [13] by Sarangi et al. This work takes into account both systematic and random intra-die variations, and apply the model to estimate timing error rates for pipeline stages in a processor with variations. Unlike this research, our work studies the effects of random intra-die variations focusing on SRAM cells

5 Conclusions

In this paper we have characterized the four types of failures that can be produced in SRAM cells (read, write, access and hold) analyzing how threshold voltage variation affects the probability of failure ranging the power supply voltage from 0.7V to 0.2V, and for 16nm feature size. Reducing the voltage, the power consumption can be highly reduced but at expense of suffering a higher amount of failures and reduced effective cache capacity. Therefore, there is a tradeoff between performance and power.

Simulation results showed that access failure is the predominant type of failure. A deeper analysis has been conducted to explore the impact of those types of failures that are dependent of WL-pulse width, that is, access and write failures. The main goal behind this work is to help computer architects to take architectural decisions to deal with the aforementioned tradeoff. In this context it would be worth to know the relationship between probability of failure and pulse length. Results have shown that the access failure probability can be reduced up to 25.5% and write failure probability by 11.4% when doubling WL-pulse width. In addition such pulse width also allows to drop the voltage 0.2V without increasing the probability of failure.

Acknowledgment. This work was supported by Spanish MICINN, Consolider Programme and Plan E funds, as well as European Commission FEDER funds, under Grants CSD2006-00046 and TIN2009-14475-C04-01.

References

1. Nassif, S.: Modeling and analysis of manufacturing variations. In: IEEE Conference on Custom Integrated Circuits, pp. 223–228 (2001)
2. Hocevar, D., Cox, P., Yang, P.: Parametric yield optimization for MOS circuit blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 7(6), 645–658 (1988)
3. Bhavnagarwala, A., Tang, X., Meindl, J.: The impact of intrinsic device fluctuations on CMOS sram cell stability. *IEEE Journal of Solid-State Circuits* 36(4), 658–665 (2001)
4. Heald, R., Wang, P.: Variability in sub-100nm SRAM designs. In: IEEE/ACM International Conference on Computer Aided Design, ICCAD 2004, pp. 347–352 (November 2004)
5. Burnett, D., Erington, K., Subramanian, C., Baker, K.: Implications of fundamental threshold voltage variations for high-density SRAM and logic circuits. In: 1994 Symposium on VLSI Technology, Digest of Technical Papers, pp. 15–16 (June 1994)
6. Agarwal, A., Paul, B., Mukhopadhyay, S., Roy, K.: Process Variation in Embedded Memories: Failure Analysis and Variation Aware Architecture. *IEEE Journal of Solid-State Circuits* 40(9), 1804–1814 (2005)
7. Zhao, W., Cao, Y.: Predictive Technology Model for Nano-CMOS Design Exploration. *Journal on Emerging Technologies in Computing Systems* 3(1), 1–17 (2007)
8. Mukhopadhyay, S., Mahmoodi, H., Roy, K.: Modeling of Failure Probability and Statistical Design of SRAM Array for Yield Enhancement in Nanoscaled CMOS. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 24(12), 1859–1880 (2005)
9. Semiconductor Industries Association, International Technology Roadmap for Semiconductors (2007), <http://www.itrs.net/>
10. Kim, J., Mccartney, M., Mai, K., Falsafi, B.: Modeling sram failure rates to enable fast, dense, low-power caches. In: IEEE Workshop on Silicon Errors in Logic (March 2009)
11. Paul, S., Cai, F., Zhang, X., Bhunia, S.: Reliability-driven ECC allocation for multiple bit error resilience in processor cache. *IEEE Transactions on Computers* 60(1), 20–34 (2011)
12. Wilkerson, C., Gao, H., Alameldeen, A., Chishti, Z., Khellah, M., Lu, S.L.: Trading off cache capacity for low-voltage operation. *IEEE Micro* 29(1), 96–103 (2009)
13. Sarangi, S., Greskamp, B., Teodorescu, R., Nakano, J., Tiwari, A., Torrelas, J.: Varius: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing* 21(1), 3–13 (2008)

Task Scheduling on Manycore Processors with Home Caches

Ananya Muddukrishna, Artur Podobas, Mats Brorsson, and Vladimir Vlassov

KTH Royal Institute of Technology, Sweden

Abstract. Modern manycore processors feature a highly scalable and software-configurable cache hierarchy. For performance, manycore programmers will not only have to efficiently utilize the large number of cores but also understand and configure the cache hierarchy to suit the application. Relief from this manycore programming nightmare can be provided by task-based programming models where programmers parallelize using tasks and an architecture-specific runtime system maps tasks to cores and in addition configures the cache hierarchy. In this paper, we focus on the cache hierarchy of the Tilera TILEPro64 processor which features a software-configurable coherence waypoint called the *home cache*. We first show the runtime system performance bottleneck of scheduling tasks oblivious to the nature of home caches. We then demonstrate a technique in which the runtime system controls the assignment of home caches to memory blocks and schedules tasks to minimize home cache access penalties. Test results of our technique have shown a significant execution time performance improvement on selected benchmarks leading to the conclusion that by taking processor architecture features into account, task-based programming models can indeed provide continued performance and allow programmers to smoothly transit from the multicore to manycore era.

1 Introduction

Faced with increasing performance demands, chip manufacturers have begun to introduce manycore processors with tens to hundreds of cores in major electronics domains. Manycore processors support a large number of cores using highly scalable architectural features such as a distributed cache hierarchy, a high bandwidth on-chip network and multiple memory controllers.

Writing software which can scale to all cores of a manycore processor is a formidable task. In addition to mapping parallelism efficiently onto cores, manycore programmers must pay close attention to memory behavior and on-chip communication of their applications. To further complicate the matter, manycore processors expose a variety of software-controllable architecture features which must be configured properly to achieve the best application performance.

Task-based programming models such as OpenMP, OmpSs [1], Cilk Plus and Intel TBB represent an important step towards easy parallel programming on shared memory machines. These models essentially allow the programmer to forget about threads and focus on expressing application parallelism using structures known as *tasks*. Tasks are internally scheduled on threads by a dynamic component called the runtime system.

By balancing the task load on threads, existing runtime systems have been able to provide good portable performance on several generations of multicore processors [2]. To continue with the same performance trend on manycore processors, task-based runtime systems will have to turn a variety of architecture-specific knobs and schedule tasks on threads more intelligently by considering chip-level aspects such as task data affinity, thread binding, cache communication latency and memory controller bandwidth.

In this paper, we report our efforts to improve task-based runtime system performance on the Tilera TILEPro64 manycore processor which features a banked chip-wide distributed software-configurable L2 cache. The cache coherence protocol of the TILEPro64 orchestrates coherence actions for each cache block from a specific bank of the L2 cache known as the *home cache*. Depending upon the location of the home cache, the access latency experienced by cores for a missing cache block can vary. We first characterize the home cache dependent non-uniformity in access times to cache blocks. Next we show how home cache oblivious task scheduling by the runtime system incurs significant execution time performance degradation. We then present a home cache aware task scheduling technique in which the runtime system implicitly controls the home cache affinity of cache blocks and schedules tasks to minimize home cache access penalties. Finally we present and explain test results of our home cache aware scheduling technique which show a significant improvement in execution time performance in comparison to blindly load-balancing runtime systems.

2 Manycore Architectures with Home Caches

In this section we introduce manycore architectural features by using the TILEPro64 as an example. In particular, we highlight the home cache feature and its impact on task scheduling performance.

2.1 TILEPro64 Processor Architecture

The Tilera TILEPro64 is a 64-core tiled architecture processor whose tiles are connected by a 8X8 multi-link mesh on-chip network. Each tile contains a 32-bit VLIW integer core, a network switch, a private 16 KB L1I cache, a private 8 KB DL1 cache and a 64 KB bank of the shared L2 cache whose aggregated capacity from 64 tiles is 4 MB. The topology of the TILEPro64 processor is shown in Figure 1a. In the topology illustration, L1 caches are considered to be part of the core and not shown. The grayed sections are explained in a later section of the paper.

The TILEPro64 provides hardware cache coherence whose actions are configurable by user-level software. The cache coherence protocol allocates every cache block sized chunk of main memory in a specific bank of the L2 cache known as the *home cache*. For simplicity, we refer to the tile that contains the home cache of a memory block as the *home tile*, and those that do not as *remote tiles*. The home cache is used to satisfy load and store requests from all tiles. Upon a load miss in the L2 cache of a tile, the home cache is requested to provide the missing block. Depending on the software configuration, the block delivered by the home cache is allocated selectively in both or either of the L2 and L1 caches of the tile. Stores in a tile are always write-through to the home cache, with a store update if the block is found in the L1.

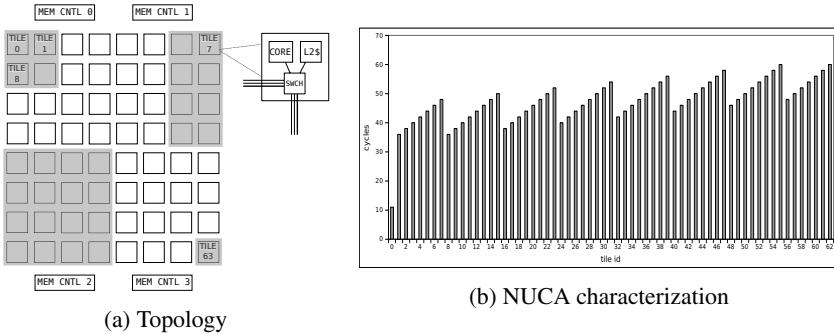


Fig. 1. TILEPro64 features

The home cache of any given cache block is selectable by software. The TILEPro64 provides the *hash-for-home* allocation scheme using which cache blocks in a main memory page are uniformly distributed on a system-wide set of home caches [3]. As an alternative, all cache blocks within a main memory page can be allocated in a single home cache. In addition, it is possible to change the home cache of a given cache block at a high migration cost.

The load latency of a missing cache block depends on the physical on-die location of the home cache. This asymmetry in cache load latency represents the Non-Uniform Cache Access (NUCA) nature of the TILEPro64 and is characterized in Figure 1b for a cache block whose home cache is in tile id 0. We can see that accesses from tiles other than tile id 0 (remote tiles) incur a 4 to 6 times increase in load latencies. We note the NUCA latencies shown in Figure 1b are measured for isolated tasks under minimal interference conditions. The actual observed NUCA latencies are indeed worse due to multi-programming, OS and hypervisor interference. We also note the NUCA latencies shown in Figure 1b are measured for tile ids up to 62 only. This is because one tile on the TILEPro64 is given up to run dedicated system software.

2.2 Home Cache Performance Impact

We now illustrate the performance degradation of home cache oblivious task scheduling using a synthetic program shown in Listing 1.1. The synthetic program is written using the OmpSs programming model which extends OpenMP with support for implicit synchronization of tasks using array-style data dependence annotations called `in`, `out` and `inout`. The synthetic program first allocates and initializes `N` blocks of data, each of size `SZ` using plain GLIBC `malloc` which internally uses the hash-for-home scheme by default on the TILEPro64. `N` tasks are then created to independently apply the transformation function, `transform`, on the `N` data blocks. We consider the performance of two different schedules for tasks of the synthetic program. The first schedule is a commonly used central queue schedule called the Breadth First Schedule (BFS). BFS represents a self-scheduled execution where newly created tasks are added to a central and made available to idle threads for execution. The second schedule is a manual schedule which is hard-coded by the programmer such that each data block has

an unique home cache and each task executes on the home tile of its data block. For N=8 and SZ=32 KB, Figures 2a and 2b respectively show the per core execution time and data cache stall cycle performance of the two schedules. We can clearly see that execution time of the home cache oblivious BF schedule suffers due to non-uniform data cache stall cycles resulting from the hash-for-home allocation of data blocks. In comparison, the manual home cache aware schedule outperforms BFS since all tasks benefit from local access to the home cache of data blocks.

```

for( int i=0; i<N; i++) {
    list [ i ] = malloc( sizeof( int ) * SZ );
    initialize ( i, list [ i ], SZ );
}
for( int i=0; i<N; i++) {
#pragma omp task inout( list [ i ][0:SZ-1] )
transform( list [ i ], SZ );
}
#pragma omp taskwait

```

Listing 1.1. Synthetic program to illustrate impact of home caches

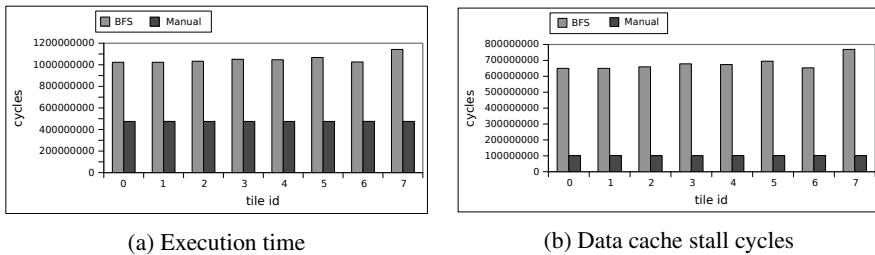


Fig. 2. Synthetic program schedule performance

The manual schedule has an obvious drawback - it requires the programmer to use TILEPro64 specific routines to explicitly set the affinity of application memory and tasks. Such an explicit responsibility is counter-productive to the programmer-friendly goal of task-based programming models which expect the programmer to only define tasks. In order to benefit from home cache aware scheduling and yet retain the programmer-friendliness of task-based programming, we decided that home cache assignment and home cache affinity based task scheduling had to become implicit runtime system responsibilities. Our implemented of the same is described in the next section.

3 Implicit Home Cache Aware Task Scheduling

We implemented implicit home cache aware task scheduling using two abstracted runtime system functions called the *memory allocation policy* and the *scheduling policy*. The memory allocation policy controls the assignment of home caches to dynamically

allocated application memory using TILEPro64 specific system calls. In our implementation, the memory allocation policy exposes two programmer interfaces called `rt_malloc` and `rt_free` as shown in Listing 1.2. These interfaces are designed similar to GLIBC `malloc` and `free` to allow trivial translation by the programmer and or source-to-source compiler.

```
void* rt_malloc (void* ref_ptr , size_t size_in_bytes );
void* rt_free (void* ptr);
```

Listing 1.2. Memory allocation policy interfaces

We implemented two types of memory allocation policies:

1. Round allocation policy: This policy assigns a single home cache to all memory pages requested by the call to `rt_malloc`. To assign the home cache, the policy chooses from pre-configured set of home caches in a round-robin manner. If the call to `rt_malloc` has a valid `ref_ptr` argument, the round allocation policy assigns the home cache of data pointed by `ref_ptr` to the currently requested allocation.
2. Hashed allocation policy: This policy uses hash-for-home allocation over a pre-configured set of home caches for all memory pages requested by the call to `rt_malloc`. The policy overrides the default system-wide set of home caches used by hash-for-home. The policy ignores the `ref_ptr` argument of `rt_malloc`.

The type of memory allocation policy is selected by a person or software that we call the *tuner*. We assume that the tuner is responsible for configuring the startup parameters of the runtime system. The tuner also decides the set of home caches used by the round and hashed policies. The memory allocation policy maintains a record of all allocations made using `rt_malloc`. Information from this record can be queried from other components of the runtime system components such as the scheduling policy.

The other half of our home cache aware scheduling mechanism is the scheduling policy called NUCA Schedule (NUCAS). During startup, NUCAS binds threads to cores using a 1:1 map. Therefore while describing NUCAS, we use the terms thread and core interchangeably. NUCAS schedules each task on the core which has the least latency of access to the home caches assigned to the dependences of the task. NUCAS determines the home caches of task dependences by querying the records of the memory allocation policy. Access latencies to task dependences are computed using a core-to-home-cache communication cost matrix whose entries are populated by calculating on-chip network latencies during runtime system initialization. Indeed, the NUCA characterization in Figure 1b graphically depicts the first row of the core-to-home-cache communication cost matrix used by NUCAS. Since the write buffer on the TILEPro64 absorbs store latencies to home caches, we designed NUCAS to consider only input dependences of tasks.

Given a task with multiple input dependences whose home caches are spread out, NUCAS checks the access latency of each core to the home caches of all task dependences and chooses the core with the least access latency. To speed up the checking process, NUCAS uses a simple heuristic for tasks with D equal sized dependences.

The heuristic simply chooses the core associated with the home tile of the last dependence in the list of D dependences. The heuristic is reasonable since scheduling a task for local access to the home cache of a single dependence on the TILEPro64 is latency-wise comparable to scheduling the same task for non-local but least latency access to home caches of all dependences. The choice of D is left to the tuner.

NUCAS associates a task queue with every core part of the runtime execution. To balance the load on the task queues, cores are grouped into fixed size vicinities and are allowed to steal tasks from other cores belonging to the same vicinity. The idea behind vicinity-based task stealing is that access latencies to home caches within a vicinity are logically considered to be the same. Vicinity sizes of 1, 4, 8, 32 and 63 tiles are made available to the tuner and their arrangement is shown in grayed sections of Figure 1a. A vicinity size of 1 implies cores never steal and a size of 63 implies all cores steal from each other. In our implementation, the default vicinity size is 1.

4 Experimental Setup

Since the TILEPro64 has many configurable knobs, we made the following assumptions to fix the architecture. We assumed that all loads and store requests issued by cores are processed by home caches only. To realize the assumption, we disabled local L1 and L2 caching on the TILEPro64. By disabling local L2 caching, we minimized the adverse effect of evicting local L2 cache entries to memory. By disabling L1 caching, we brought the NUCA impact of home caches out to the front. This move additionally allowed us to make a technology projection to home cache based architectures larger and slower than that of the TILEPro64.

We used four task-based applications as benchmarks to test the performance of NU CAS. Two of the benchmarks are synthetic but with real world execution patterns. The benchmarks are described below:

1. Map: This is a synthetic benchmark whose execution resembles the map phase of the Map-Reduce programming framework. The benchmark allocates data in chunks and creates tasks which independently operate on unique data chunks. The benchmark is exactly similar to the synthetic program shown in Listing 1.1.
2. Aggregator: This is a synthetic benchmark whose execution resembles the merge phase of the Mergesort benchmark of the Barcelona OpenMP Task Suite (BOTS) [4]. The benchmark's task dependence graph begins with a execution similar to the Map benchmark and later unfolds as an inverted tree. Each task in the inverted tree reads from two to three input dependences, operates on the read data and writes results to a single output dependence, thereby performing a reduction operation.
3. Vector Multiplier: This benchmark uses tasks to perform vector transformations commonly seen in scientific applications. Each task of the benchmark iteratively applies a set of multiply and add operations on two input vectors and stores back the result into one of the input vectors. The input vectors are not co-allocated in the same home cache. Tasks of the benchmark are independent, therefore the dependence graph of the benchmark is flat.

4. SparseLU: This is a benchmark based on the SparseLU benchmark of BOTS. This benchmark uses tasks to perform the LU factorization of the input sparse matrix. Each task of the benchmark reads from two to three blocks of the sparse matrix and stores back the result into one of the blocks. Two classes of tasks access one of the input blocks more intensely than others. Tasks of the benchmark are not independent and the task dependence graph is complex.

While selecting benchmark parameters, we had to ensure that the NUCA impact of home caches was not masked by other architectural features of the TILEPro64. We considered integer versions of the benchmarks to rule out effects of slow floating-point operations which on the TILEPro64 are emulated using software. The benchmark data sizes were carefully selected to minimize off-chip memory accesses which take about 180 cycles on the TILEPro64. In addition, we zeroed the interval of the operating system tick scheduler to minimize thread switching effects. Finally, the benchmarks were run in isolation to avoid cache pollution effects from other applications. The benchmark input parameters and related information are summarized in Table 1.

Table 1. Study benchmark parameters

Benchmark	Input	Tuner Input	Number of tasks
Map	63 chunks, each 16 KB	Cores=63, D=1	63
Aggregator	48 chunks, each 16 KB	Cores=48, D=3	94
Vector Multiplier	128 integer vectors, each 4096 integers	Cores=63, D=2	4096
SparseLU	32X32 blocks, each 36X36 integers	Cores=63, D=3	3281

We implemented NUCAS and memory allocation policies using an in-house experimental task-based runtime system called MIR. MIR supports task scheduling and implicit task synchronization. In addition, MIR records core execution states and hardware performance counter events and dumps trace files viewable in Paraver [5] which is a powerful parallel execution visualization tool developed by the Barcelona Supercomputing Center (BSC).

To compare NUCAS, we used the BFS schedule. The execution of BFS is based on a similarly named scheduling policy described in a study of OpenMP task scheduling strategies by Duran et al [6]. BFS associates a single FIFO task queue with all cores part of the runtime execution. BFS queues all application tasks into the single queue. When idle, each core removes and executes the task from the FIFO queue. Due to this random self-scheduling nature, BFS is fast and provides aggressive load-balancing for medium-grained tasks. We did not implement a distributed queue scheduler for comparison since our study benchmarks have medium grained tasks all created by a single thread [2]. We also stress that we do not intend to make an apples-to-apples comparison of scheduling policies. The idea behind our choice of comparison was to judge the performance of NUCAS which optimized for home caches against BFS which did not.

We selected the following combinations of scheduling and memory allocation policies: BFS-hashed, BFS-round and NUCAS-round. The BFS-hashed policy can be considered as the only option available to the tuner on the TILEPro64 in the absence of the home cache aware scheduling mechanisms. We consider the BFS-round case as an interesting interconnect bandwidth optimizing experiment in which the programmer explicitly allocates application data on different home caches but the scheduler is oblivious to the distribution and schedules tasks randomly. Finally in the NUCAS-round case, we characterize a runtime system execution where application data is implicitly allocated on different home caches and passed on as useful information to the scheduler which in turn schedules tasks to minimize NUCA penalties of home caches. We do not consider the NUCAS-hashed case since it unfeasibly involves minimizing NUCA penalties to cache blocks spread out on all available home caches. To add, system software of the TILEPro64 does not provide an user-level interface to obtain the home cache mapping on a cache block basis for hash-for-home allocated data.

5 Experimental Results

Figure 3 shows the execution time performance of our study benchmarks under different scheduling and memory allocation policy combinations which we simply refer to as *schedules*. The time measured corresponds to the parallel section of the benchmarks. For the Vector Multiplier and SparseLU benchmarks, performance of task stealing under different vicinity sizes is also shown. Since the Map and Aggregator benchmarks do not improve with vicinity-based task stealing, we only show the NUCAS-round result for a vicinity size of 1 for these benchmarks. Figure 3 clearly shows that NUCAS-round produced the fastest schedule for all the benchmarks.

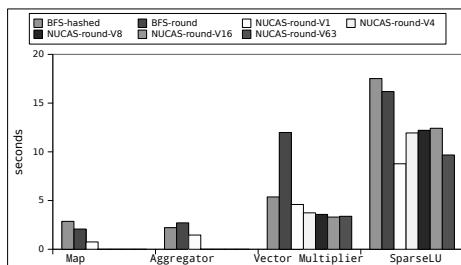


Fig. 3. Execution time performance of study benchmarks

In order to ascertain that home cache aware task scheduling, or the lack of it, was the reason behind the performance results seen in Figure 3, we observed core execution traces complemented with cycle counter and data cache stall cycle counter values on Paraver. Columns 4a- 4d in Figure 4 shows the Paraver *view* of core execution traces for all study benchmarks. The rows of Figure 4 from top to bottom respectively show the views of BFS-hashed (B-h), BFS-round (B-r) and NUCAS-round (N-r) schedules. The NUCAS-round view corresponds to the best performing vicinity size in Figure 3.

Within each view, the Y-axis marks cores and while the X-axis shows cycles. Each view shows a per core timeline, i.e., when and for how long each core executed tasks, using colored bars. The color of each bar is encoded using a gray gradient. Light and dark gray bars respectively indicate low and high data cache stall cycle values. The three schedule views of a benchmark are relative to each other - they are cycle aligned to the largest execution time and are semantic aligned to fit the entire range of data cache stall cycles values seen among all schedules.

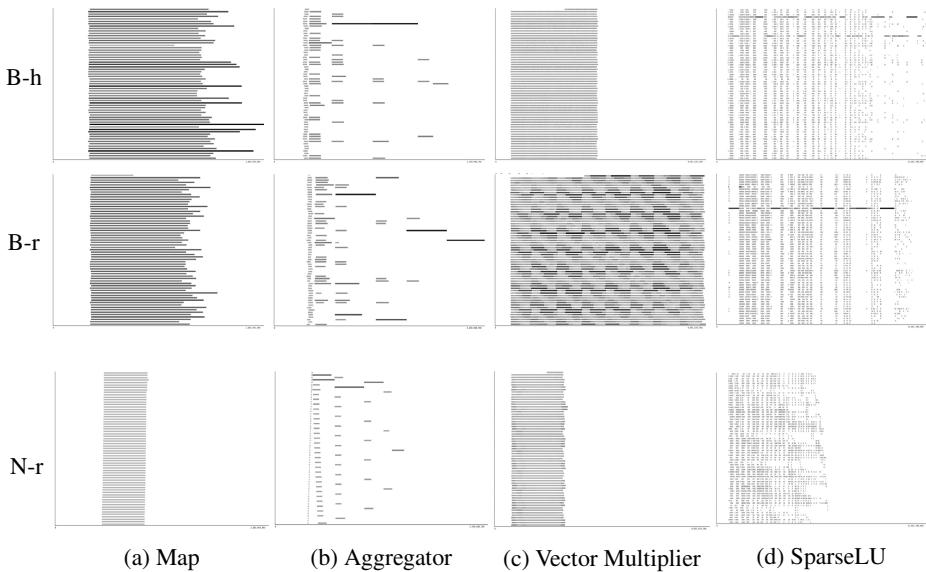


Fig. 4. Paraver views of study benchmark execution under different schedules

For the Map benchmark case, the views agreed with our motivation for home cache aware scheduling presented in Section 2.2. Views for both BFS-hashed and BFS-round schedules, which are home cache oblivious, showed long dark gray bars, indicating long task execution times due to large number of data cache stall cycles. On the other hand, the view for the home cache aware NUCAS-round schedule showed short light gray bars, indicating relatively smaller task execution times due to small number of data cache stall cycles. We reasoned about the performance of different schedules for the Aggregator benchmark similarly.

The BFS-round schedule performed poorly for the Vector Multiplier benchmark showing a large number of dark gray bars in its view. In comparison, views of both the BFS-hashed and NUCAS-round showed only light gray bars. However, the NUCAS-round schedule surpassed the performance of BFS-hashed schedule showing relatively lighter gray bars indicating that tasks suffered lower data cache stall cycles in comparison. In addition, vicinity-based task stealing improved execution time performance due to load-balancing over larger vicinity sizes.

The views of the SparseLU benchmark presented an interesting case. Both the BFS-round and BFS-hashed schedules showed that almost all tasks save a few experienced uniform data cache stall cycles and short execution times. However, those few tasks which incurred large data cache stall cycles (dark gray bars) increased the critical path of execution. In comparison, the NUCAS-round view showed that all tasks experienced uniform data cache style cycles the reason for which is the NUCAS heuristic which was able to optimize task execution for local access to the home cache of the most intensely accessed dependence. Vicinity-based stealing, which is oblivious to dependence access intensity, effectively removed the benefit of heuristic.

6 Related Work

Task and data affinity mechanisms discussed in our work are greatly inspired by the large body of research on NUMA optimizations for OpenMP runtime systems. The implicit memory allocation and architectural locality based scheduling mechanisms we implemented in the runtime system are inspired by a similar work on NUMA systems by Broquedis et al [7]. The memory allocation policies in our work are similar in principle to the *bind* and *cyclic* policies of the Minas memory allocator framework [8] for large scale NUMA machines. We cannot compare the techniques of our study with NUMA-based research because of the large difference in permissible scheduling overheads - NUCA penalties cost tens of cycles whereas NUMA penalties cost thousands of cycles. Our work improves on random task scheduling by reducing the impact of cache access penalties. We found a similar motivation in the SLAW [9] and the MTS [10] work-stealing schedulers for machines with hierarchical memory hierarchies. Both SLAW and MTS are designed to restrict off-chip work-stealing and execute parent and sibling tasks on the same multicore processor to utilize the state maintained in the central last-level cache. Our work relies on simple memory allocation policies to distribute task data on home caches during run-time. We found a more complicated counterpart in the work of Lu et al [11] where loops of affine programs are transformed at compile-time to minimize the NUCA impact of hash-for-home style of allocation manycore processors.

7 Conclusions

Manycore processors have hit the market and will grow larger in size and complexity in the coming years. In our work, we have shown that manycore processors demand careful architecture configuration and scheduling to achieve scalable parallel performance. Using the performance impact of home caches on the Tilera TILEPro64 processor as an example, we have shown how the runtime system of task based programming models can implicitly configure the manycore architecture for improved task scheduling performance. We believe that techniques similar to our memory allocation policy design, which essentially pushes application memory management as a runtime system responsibility, will rise in prominence as manycore processors become commonplace. Our home cache aware scheduling technique has shown significant execution time improvement in comparison to plain aggressive load-balancing schedules for selected study benchmarks. Reliance on the tuner, chunked application memory allocation, fixed steal

vicinity size, static memory allocation policy, unknown trigger points to recalculate the core-to-home-cache communication cost matrix and a limited set of study benchmarks constitute the currently being addressed limitations of our work. Scheduling is a game of overheads and with the arrival of manycore processors, we have shown that task-based runtime systems can and should begin thinking about what really goes on inside the processor.

Acknowledgment. The research leading to these results has received funding from the European Commission's Seventh Framework Programme [FP7/2007-2013] under the ENCORE Project (www.entrepreneur-project.eu), grant agreement nr. 248647. The authors are members of the HiPEAC European network of Excellence (<http://www.hipeac.net>). The authors acknowledge great support in understanding Paraver from Xavi Aguilar at PDC Center For High Performance Computing at KTH, Xavier Teruel and Alejandro Rico Carro, both at BSC.

References

1. Duran, A., Ayguad, E., Badia, R., Labarta, J., Martinell, L., Martorell, X., Planas, J.: Ompss: a proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters* 21(2), 173 (2011)
2. Podobas, A., Brorsson, M.: A comparison of some recent task-based parallel programming models. In: Proceedings of the 3rd Workshop on Programmability Issues for Multi-Core Computers, MULTIPROG 2010, Pisa (January 2010)
3. Tilera: Tile processor user architecture manual, <http://www.tilera.com/scm/docs/UG101-User-Architecture-Reference.pdf> (accessed June 14, 2012)
4. Duran, A., Teruel, X., Ferrer, R., Martorell, X., Ayguade, E.: Barcelona openmp tasks suite: A set of benchmarks targeting the exploitation of task parallelism in openmp. In: International Conference on Parallel Processing, ICPP 2009, pp. 124–131. IEEE (2009)
5. Pillet, V., Labarta, J., Cortes, T., Girona, S.: Paraver: A tool to visualize and analyze parallel code. In: WoTUG-18, pp. 17–31 (1995)
6. Duran, A., Corbalán, J., Ayguadé, E.: Evaluation of OpenMP Task Scheduling Strategies. In: Eigenmann, R., de Supinski, B.R. (eds.) IWOMP 2008. LNCS, vol. 5004, pp. 100–110. Springer, Heidelberg (2008)
7. Broquedis, F., Furmento, N., Goglin, B., Namyst, R., Wacrenier, P.A.: Dynamic Task and Data Placement over NUMA Architectures: An OpenMP Runtime Perspective. In: Müller, M.S., de Supinski, B.R., Chapman, B.M. (eds.) IWOMP 2009. LNCS, vol. 5568, pp. 79–92. Springer, Heidelberg (2009)
8. Pousa Ribeiro, C., Méhaut, J.F.: Minas: Memory Affinity Management Framework. Research Report RR-7051, INRIA (2009)
9. Guo, Y., Zhao, J., Cave, V., Sarkar, V.: Slaw: a scalable locality-aware adaptive work-stealing scheduler. In: 2010 IEEE International Symposium on Parallel & Distributed Processing, IPDPS, pp. 1–12. IEEE (2010)
10. Olivier, S., Porterfield, A., Wheeler, K., Prins, J.: Scheduling task parallelism on multi-socket multicore systems. In: Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers, pp. 49–56. ACM (2011)
11. Lu, Q., Alias, C., Bondhugula, U., Henretty, T., Krishnamoorthy, S., Ramanujam, J., Rountev, A., Sadayappan, P., Chen, Y., Lin, H., et al.: Data layout transformation for enhancing data locality on nuca chip multiprocessors. In: 18th International Conference on Parallel Architectures and Compilation Techniques, PACT 2009, pp. 348–357. IEEE (2009)

ParaPhrase Workshop 2012

M. Danelutto¹, K. Hammond², and H. Gonzalez-Velez³

¹ Pisa University (I)

² St. Andrews University (UK)

³ NCI Cloud Competency Centre (IRL)

Foreword

ParaPhrase (*Parallel Patterns for Adaptive Heterogeneous Multicore Systems*) is a three year FP7 EU funded project that started in October 2011¹. The project aims to develop and deploy new high-level design patterns for parallel applications that support alternative parallel implementations that can be initially mapped and subsequently dynamically re-mapped to the available *heterogeneous* (CPU+GPU) hardware. The ParaPhrase approach leverages a two-level (or ultimately multi-level) model of parallelism, where the implementations of parallel programs are expressed in terms of interacting components, and where components from different applications are collectively mapped to the available system resources. By expressing parallelism in terms of high-level parallel patterns that have alternative parallel implementations, ParaPhrase tools are able to re-deploy/refactor parallel components to match the available hardware resources. *Refactoring* (programmer-directed source code transformation) is assumed to take place within ParaPhrase both at an application level and at a complete systems level, involving multiple parallel workflows running on a heterogeneous collection of CPU/GPU execution units. Moreover, by using a strong component basis for parallelism, ParaPhrase tools are expected to achieve potentially significant gains in terms of enhancing sharing at a high level of abstraction, and so in reducing or even eliminating the costs that are usually associated with cache management, locking, and synchronization.

ParaPhrase organized the 2012 ParaPhrase workshop, co-located with EuroPar in Rhodes Island, Greece, as its second open ParaPhrase workshop, following on from the successful HPLGPU 2012 on High-Level Programming for Heterogeneous and Hierarchical Parallel Systems (January 2012, co-located with HiPEAC, Paris).

The 2012 ParaPhrase workshop aimed to bring together researchers working on the various research issues covered by ParaPhrase and to discuss achievements, open research problems and perspectives in the area. Topics of interest included: Advanced parallel programming models, Methods for targeting Heterogeneous parallel architectures, Refactoring techniques for parallelism, Algorithmic skeletons, Parallel design patterns, Parallelism in functional programming

¹ <http://www.paraphrase-ict.eu>

languages, Formal tools for parallel programming, Efficient mechanisms supporting multi-/many-core programming, Deployment and configuration tools for parallel heterogeneous architectures.

In addition to unrefereed project-specific talks, including a general overview by the project coordinator (Kevin Hammond, St Andrews University), the ParaPhrase workshop included the two refereed papers which are published here. The first, *Using the SkelCL Library for High-Level GPU Programming of 2D Applications* by M. Steuwer, S. Gorlatch, M. Buß, and S. Breuer discusses the implementation of particular applications using the algorithmic skeleton library SkelCL, offering pre-implemented recurring computation and communication patterns (skeletons) which greatly simplify programming for single- and multi-GPU systems. The second, *Structured data access annotations for massively parallel computations* by M. Aldinucci, S. Campa, P. Kilpatrick and M. Torquati, introduces a methodology aimed at supporting the joint exploitation of control (stream) and data parallelism in algorithmic skeleton based parallel programming environments. Both of these papers focus on research topics that are central to ParaPhrase research activities. Overall, the workshop ran smoothly, with a good number of attendees and plenty of discussion. We are grateful to the Euro-Par organizing team for their ongoing support, publicity, and excellent choice of location, as well as to the contributing authors, who helped to make this early workshop a success.

We are just at the start of a new period of increasing architectural heterogeneity. The ParaPhrase approach offers a way of abstracting/virtualising this heterogeneity so that much of the complexity can be managed at design/implementation time, and so that parallel applications are resilient to future architectural changes. We look forward to an exciting period of research dealing with the challenges that will arise from this exciting new research area.

Using the SkelCL Library for High-Level GPU Programming of 2D Applications

Michel Steuwer, Sergei Gorlatch, Matthias Buß, and Stefan Breuer

University of Münster, Münster, Germany
`{michel.steuwer,gorlatch}@uni-muenster.de`

Abstract. Application programming for GPUs (Graphics Processing Units) is complex and error-prone, because the popular approaches — CUDA and OpenCL — are intrinsically low-level and offer no special support for systems consisting of multiple GPUs. The SkelCL library offers pre-implemented recurring computation and communication patterns (*skeletons*) which greatly simplify programming for single- and multi-GPU systems. In this paper, we focus on applications that work on two-dimensional data. We extend SkelCL by the matrix data type and the MapOverlap skeleton which specifies computations that depend on neighboring elements in a matrix. The abstract data types and a high-level data (re)distribution mechanism of SkelCL shield the programmer from the low-level data transfers between the system’s main memory and multiple GPUs. We demonstrate how the extended SkelCL is used to implement real-world image processing applications on two-dimensional data. We show that both from a productivity and a performance point of view it is beneficial to use the high-level abstractions of SkelCL.

1 Introduction

Application programming for GPUs (Graphics Processing Units) is complex and error-prone. The popular programming models for systems with GPUs – CUDA and OpenCL [2,4,5] – require the programmer to explicitly manage GPU’s memory, including (de)allocations and data transfers to/from the system’s main memory. This leads to lengthy, low-level, complex and thus error-prone code. For emerging systems with multiple GPUs, CUDA and OpenCL additionally require an explicit implementation of data exchange between GPUs and separate management of each GPU. This includes low-level pointer arithmetics and offset calculations, as well as explicit program execution on each GPU. Neither CUDA nor OpenCL offer specific support for such systems, which makes their programming even more complex.

In this paper, we first briefly describe our SkelCL library [9] for high-level single- and multi-GPU computing. It offer pre-implemented recurring computation patterns (*skeletons*) for simplified GPU programming. In addition, the application developer is freed from memory management, which is done implicitly in SkelCL.

As a specific contribution of the paper, we add a new two-dimensional data type and an additional skeleton to SkelCL. The *matrix* data type complements the one-dimensional vector data type for working with two-dimensional data, e.g., matrices or images. The new *MapOverlap* skeleton executes a given function on every element of the input data, using also the values of its neighboring elements. Another novel contribution of this paper is the data distribution mechanism for simplifying two-dimensional data processing on systems with multiple GPUs. Finally, we present an application case study using the matrix data type and the *MapOverlap* skeleton – Sobel edge detection for 2D images – and report experimental results using SkelCL.

The paper is organized as follows. First we briefly introduce the basics of SkelCL in Section 2 using our previous work [9]. The *MapOverlap* skeleton is then introduced in Section 2.2 and the matrix data type in Section 2.3. In Section 3 we demonstrate how both are used together to implement an application from the area of image processing: the Sobel edge detection. We report experimental results and we evaluate performance and usability of our approach. Section 4 concludes the paper and compares our approach to related work.

2 SkelCL

We have designed SkelCL as a library for high-level programming of multi-core CPU, GPU, and other processing units. SkelCL is built on top of OpenCL and provides a C++ API that shields the programmer from boilerplate code, e.g., for program initialization or recurring tasks such as explicit data transfers between CPU and GPU. Programming is simplified using algorithmic skeletons – generic building blocks that describe commonly used parallel computation and communication patterns. For flexibility, SkelCL can also be used in combination with low-level OpenCL code. The SkelCL library is available as open-source software and can be downloaded from: <http://skelcl.uni-muenster.de>.

2.1 Algorithmic Skeletons

In standard OpenCL parallelism is specified using, special functions (*kernels*) to be executed in a parallel manner on a device. It is programmer’s task to specify in the host program how many instances of a kernel are launched. In addition, kernels usually take pointers to device memory as input and contain program code for reading/writing single data items from/to it. These pointers have to be used carefully, because no boundary checks are performed by OpenCL.

To shield the programmer from these low-level programming tasks, the SkelCL library extends OpenCL by means of high-level programming patterns, called *algorithmic skeletons*. Formally, a skeleton is a higher-order function that executes one or more user-defined functions in a pre-defined parallel manner, while hiding the details of parallelism and communication from the user [8].

Four basic skeletons are provided by SkelCL: *Map*, *Zip*, *Reduce*, and *Scan*. We describe these skeletons semi-formally, with v_{in} , v_{inl} , v_{inr} , and v_{out} denoting vectors of size n , with $0 \leq i < n$:

```

int main (int argc, char const* argv[]) {
    SkelCL::init();                                /* initialize SkelCL */
    Reduce<float> sum (                           /* create skeletons */
        "float func(float x, float y) { return x+y; }" );
    Zip<float> mult (
        "float func(float x, float y) { return x*y; }" );
                                                /* create input vectors */
    Vector<float> A(SIZE); fillVector(A);
    Vector<float> B(SIZE); fillVector(B);
                                                /* execute skeletons */
    Vector<float> C = sum( mult( A, B ) );
    cout << "Result: " << C.front(); /* print result */
}

```

Listing 1.1. SkelCL program computing the dot product of two vectors

- The *Map* skeleton applies a unary function f to each element of an input vector v_{in} , i. e. $v_{out}[i] = f(v_{in}[i])$.
- The *Zip* skeleton operates on two input vectors v_{inl} and v_{inr} , applying a binary operator \oplus to all pairs of elements, i. e. $v_{out}[i] = v_{inl}[i] \oplus v_{inr}[i]$.
- The *Reduce* skeleton computes a scalar value r from a vector using a binary operator \oplus , i. e. $r = v[0] \oplus v[1] \oplus \dots \oplus v[n - 1]$.
- The *Scan* skeleton (a. k. a. prefix-sum) yields an output vector with each element obtained by applying a binary operator \oplus to the elements of the input vector up to the current element’s index, i. e. $v_{out}[i] = \bigoplus_{j=0}^{i-1} v_{in}[j]$.

Rather than writing low-level kernels, in SkelCL the programmer customizes skeletons by providing user-defined (usually less complex) functions.

Listing 1.1 shows how a dot product of two vectors is implemented in SkelCL using two skeletons: the *Zip* skeleton is customized by usual multiplication, and the *Reduce* skeleton is customized by usual addition. This program comprises 8 lines of code (omitting comments and empty lines). For comparison, an OpenCL-based implementation of a dot product provided in the NVIDIA SDK [1] requires 68 lines of code (kernel function: 9 lines, host program: 59 lines). Besides, additional code would be necessary for a multi-device implementation, including statements for data transfer between multiple devices and for splitting input data and merging output data on the host.

In SkelCL, skeletons can be executed on single- and multi-device systems. In case of a multi-device system, the calculation specified by a skeleton is performed automatically on all devices available to the system. The SkelCL program in Listing 1.1 can thus be executed on a multi-device system without any change.

2.2 The MapOverlap Skeleton

Many applications dealing with two-dimensional data perform calculations for every data element taking neighboring data elements into account. For example, image processing algorithms, like the gaussian blur, calculate a new value

for every pixel of an input image using the previous value of the pixel and its surrounding values.

To facilitate the development of such applications, we extend SkelCL by an additional skeleton in combination with a new matrix data type, which is presented in Section 2.3. This skeleton can be used with either vector or matrix data type. We explain the details of the new skeleton for the matrix data type.

- The *MapOverlap* skeleton takes two parameters: a function f and an integer value d . It applies f to each element of an input matrix m_{in} while taking the neighboring elements within the range $[-d, +d]$ in each dimension into account, i. e.

$$m_{out}[i, j] = f \begin{pmatrix} m_{in}[i - d, j - d] & \dots & m_{in}[i - d, j] & \dots & m_{in}[i - d, j + d] \\ \vdots & & \vdots & & \vdots \\ m_{in}[i, j - d] & \dots & m_{in}[i, j] & \dots & m_{in}[i, j + d] \\ \vdots & & \vdots & & \vdots \\ m_{in}[i + d, j - d] & \dots & m_{in}[i + d, j] & \dots & m_{in}[i + d, j + d] \end{pmatrix}$$

In the actual source code, the application developer provides the function f which receives a pointer to the element in the middle, $m_{in}[i, j]$. Listing 1.2 shows a simple example of computing the sum of all direct neighboring values using the *MapOverlap* skeleton. To access the elements of the input matrix m_{in} , function *get* is used, as provided by SkelCL. All indices are specified relative to the middle element $m_{in}[i, j]$, therefore, for accessing this element the function call `get(m_in, 0, 0)` is used.

The application developer must ensure that only elements in the range specified by the second argument d of the *MapOverlap* skeleton, are accessed. In Listing 1.2, range is specified as $d = 1$, therefore, only direct neighboring elements are accessed. To enforce this property, boundary checks are performed at runtime by the *get* function. In future work, we plan to avoid boundary checks at runtime by statically proving that all memory accesses are in bounds, as it is the case in the shown example.

Special handling is necessary when accessing elements out of the boundaries of the matrix, e.g., when the item in the top-left corner of the matrix accesses elements above and left of it. The *MapOverlap* skeleton can be configured to handle such out-of-bound memory accesses in two possible ways: 1) a specified neutral value is returned; 2) the nearest valid value inside the matrix is returned. In Listing 1.2, the first option is chosen and 0.0 is provided as neutral value.

Listing 1.3 shows how the same simple calculation can be performed in standard OpenCL. While the amount of lines of code increases by a factor of 2, the complexity of each single line also increases, as follows. Besides a pointer to the output memory, the width of the matrix has to be provided as parameter. The correct index has to be calculated for every memory access using an offset and the width of the matrix. Therefore, knowledge about how the two-dimensional matrix is stored in one-dimensional memory is required. In addition, manual boundary checks have to be performed to avoid faulty memory accesses.

```
MapOverlap<float(float)> m("float func(float* m_in) {
    float sum = 0.0f;
    for (int i = -1; i < 1; ++i)
        for (int j = -1; j < 1; ++j)
            sum += get(m_in, i, j);
    return sum;
}", 1, SCL_NEUTRAL, 0.0f);
```

Listing 1.2. MapOverlap skeleton computing the sum of all direct neighbors for every element in a matrix

```
__kernel void sum_up(__global float* m_in,
                     __global float* m_out,
                     int width, int height) {
    int i_off = get_global_id(0); int j_off = get_global_id(1);
    float sum = 0.0f;
    for (int i = i_off - 1; i < i_off + 1; ++i)
        for (int j = j_off - 1; j < j_off + 1; ++j) {
            // perform boundary checks
            if (i < 0 || i > width || j < 0 || j > height)
                continue;
            sum += m_in[j * width + i];
        }
    m_out[j_off * width + i_off] = sum; }
```

Listing 1.3. An OpenCL kernel performing the same calculation as the MapOverlap skeleton shown in Listing 1.2

SkelCL avoids all these low-level details. Neither additional parameter, nor index calculations or manual boundary checks are necessary. In SkelCL, the application developer only provides the source code implementing the steps required by the algorithm.

2.3 Abstract Data Types and Memory Management

The extended SkelCL offers two *containers*: the vector and the matrix. While the vector offers support for one-dimensional data, the new matrix data type handles two-dimensional data.

Vector data type. The *vector* class provides an abstraction for a contiguous memory area that is accessible by both, the host and the device. The SkelCL vector replicates the interface of the vector from the Standard Template Library (STL), i.e., it can be used as a replacement of the standard vector. Upon creation of a vector on the host system, memory is allocated on the device accordingly.

The vector class shields the user from low-level memory operations like allocation (on the device) and data transfer between host and device memory, as follows. Data transfers between the memory areas on the host and device are performed implicitly: before any data is accessed on the host, SkelCL ensures

that the data on the host is up-to-date. This may lead to implicit data transfers from the device which are performed automatically. All skeletons can accept vectors as their input and output. Before execution, a skeleton’s implementation ensures that all input vectors’ data is available on all participating devices. This may result in implicit (automatic) data transfers from the host memory to device memory. The data of the output vector is not copied back to the host memory but rather resides in the device memory. Hence, if an output vector is used as the input to another skeleton, no further data transfer is performed. This *lazy copying* in SkelCL minimizes costly data transfers between the host and device.

Matrix data type. In addition to the vector as a one-dimensional abstract data structure, we introduce in SkelCL a two-dimensional abstract data type, the *matrix*. Developing applications on 2D data for modern parallel architectures is cumbersome, since efficient memory handling is required for achieving good performance. In case of GPUs, exploiting the memory hierarchy by using the fast but small on-chip memory is mandatory for high performance. Simple and obvious solutions taken directly from a textbook will most certainly result in sub-optimal performance.

The matrix data type in SkelCL automatically manages all data transfers between the host’s memory and the devices’ memory. Necessary data transfers are performed implicitly: when a matrix is used by a skeleton, SkelCL copies the matrix’s data to the devices, and vice versa. That means that when the data of the matrix is accessed on the host, SkelCL copies the modified data back to the host. Like the vector type, the matrix shields the user from dealing with low-level details like data transfers between different memories. The Map, Zip and MapOverlap skeleton can take matrices as input and output.

2.4 Data Distribution on Multiple Devices

The key feature of SkelCL for multi-device systems is that SkelCL’s data types abstract from memory ranges on multiple devices, i. e. the data is accessible by each device. However, each device may access different parts of a container (vector or matrix) or may even not access it at all. For example, when implementing work-sharing on multiple devices, the devices will usually access disjoint parts of input data, such that copying only a part of the container to a device would be more efficient than copying the whole data to each device.

To simplify the specification of partitionings of containers in programs for multi-device systems, SkelCL implements the *distribution* mechanism that describes how a container is distributed among the available devices. It allows the programmer to abstract from managing memory ranges which are shared or spread across multiple devices: the programmer can think of a distributed container as of a self-contained entity.

Four kinds of distribution are currently implemented in SkelCL and offered to the programmer: *block*, *copy*, *single* and *overlap* (see Figure 1). With *block* distribution (Figure 1a), each device stores a contiguous, disjoint part of the container. The *copy* distribution (Figure 1b) copies container’s entire data to

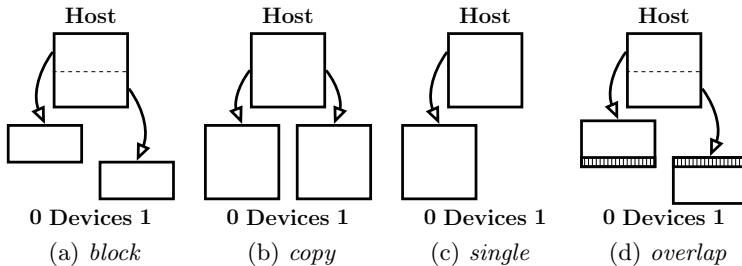


Fig. 1. Distributions of a matrix in SkelCL

each available device. In case of *single* distribution (Figure 1c), container's whole data is stored on a single device (the first device by default).

Together with the matrix data type, we introduce a new distribution called *overlap* (Figure 1d). The overlap distribution splits the matrix into one chunk for each device, similarly to the block distribution. In addition to the block distribution, the following holds for an overlap-distributed matrix: each chunk consists of a number of continuous rows, and, a parameter – the *overlap size* – specifies the number of rows at the edges of a chunk which are copied to the two neighboring devices. Figure 1d illustrates the overlap distribution: Device 0 receives the top chunk ranging from the top row to the middle, while device 1 receives the second chunk ranging from the middle row to the bottom. The marked regions are the *overlap regions* which are available on both devices.

The *overlap* distribution is automatically selected as distribution by the MapOverlap skeleton, to ensure that every device has access to the neighboring elements as needed by the MapOverlap skeleton.

A programmer can set the distribution of containers explicitly, or every skeleton selects a default distribution for its input and output containers otherwise. Container's distribution can be changed at runtime. A change of distribution implies data exchanges between multiple devices and the host, which are performed by SkelCL implicitly and lazily, as described above. Implementing such data transfers in the standard OpenCL is a cumbersome task: data has to be downloaded to the host before it can be uploaded to other devices, including the corresponding length and offset calculations; this results in a lot of low-level code which is completely hidden when using SkelCL.

3 Application Study: Sobel Edge Detection

To evaluate the usability and performance of the MapOverlap skeleton and the matrix data type, we implemented an algorithm commonly used in image processing: The Sobel edge detection is applied to an input image and produces an output image, in which the detected edges in the input image are marked in white and plain areas are shown in black. Figure 2 shows the famous Lena image [7] and the output of Sobel edge detection applied to it.



(a) Original image (b) Image after Sobel edge detection

Fig. 2. The famous Lena image often used as an example in image processing

```

for (i = 0; i < width; ++i)
    for (j = 0; j < height; ++j)
        h = -1*img[i-1][j-1] +1*img[i+1][j-1]
            -2*img[i-1][j] +2*img[i+1][j]
            -1*img[i-1][j+1] +1*img[i+1][j+1];
        v = ...;
        out_img[i][j] = sqrt(h*h + v*v);
    
```

Listing 1.4. Sequential implementation of the Sobel edge detection

Listing 1.4 shows the algorithm of the Sobel edge detection in pseudo-code. To keep this version simple, necessary boundary checks are omitted. In this sequential version, for computing one output value `out_img[i][j]` the input value `img[i][j]` and the direct neighboring elements are needed. Therefore, the `MapOverlap` skeleton is a perfect fit for implementing the Sobel edge detection.

Listing 1.5 shows the SkelCL implementation using the `MapOverlap` skeleton and the matrix type. The implementation is straightforward and very similar to the sequential version in Listing 1.4. The only notable difference is that for accessing elements the `get` function is used instead of the square bracket notation.

Listing 1.6 shows a part of the standard OpenCL implementation for Sobel edge detection. The actual computation is performed inside the `computeSobel` function, which is omitted in the listing, since it is quite similar to the sequential

```

// skeleton customized with Sobel edge detection algorithm
MapOverlap<char(char)> m( "char func(const char* img) {
    short h = -1*get(img, -1, -1) +1*get(img, +1, -1)
        -2*get(img, -1, 0) +2*get(img, +1, 0)
        -1*get(img, -1, +1) +1*get(img, +1, +1);
    short v = ...;
    return sqrt(h*h + v*v); }", 1, SCL_NEUTRAL, 0);
Matrix<char> out_img = m(img); // execution of the skeleton
    
```

Listing 1.5. SkelCL implementation of the Sobel edge detection

```

__kernel void sobel_kernel( __global const char* img,
                           __global      char* out_img,
                           int w, int h ) {
    size_t i = get_global_id(0);    size_t j = get_global_id(1);

    if(i < w && j < h) {
        // perform boundary checks
        char ul = (j-1 > 0 && i-1 > 0) ? img[((j-1)*w)+(i-1)] : 0;
        char um = (j-1 > 0)           ? img[((j-1)*w)+(i+0)] : 0;
        char ur = (j-1 > 0 && i+1 < w) ? img[((j-1)*w)+(i+1)] : 0;
        // ... 5 more
        char lr = (j+1 < h && i+1 < w) ? img[((j+1)*w)+(i+1)] : 0;

        out_img[j * w + i] = computeSobel(ul, um, ur, ..., lr); } }
```

Listing 1.6. Additional boundary checks and index calculations, necessary in the standard OpenCL implementation

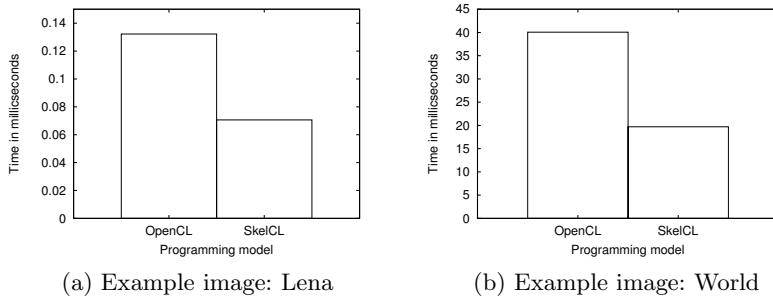


Fig. 3. Performance results (runtimes)

version describing the actual Sobel edge detection algorithm. The listing shows that extra low-level code is necessary to deal with technical details, like boundary checks and index calculations. These extra lines are arguably complex and error-prone because they handle low-level details, rather than the application logic.

We performed runtime experiments using a NVIDIA Tesla T10 GPU with 480 processing elements and 4 GByte memory. Figure 3 shows the runtime of the OpenCL version in Listing 1.6 vs. the SkelCL version with the MapOverlap skeleton in Listing 1.5. Only the kernel runtimes are shown, as the data transfer times are equal for both versions. Measurements were taken using the OpenCL profiling API. Besides the Lena image [7] with a size of 512×512 pixel, we also used a bigger image by NASA showing the world [6] with a resolution of 15296×7648 pixel. The mean values of 6 runs are shown in Figure 3. The Lena image is shown on the left, the NASA world image on the right.

The SkelCL version clearly outperforms the OpenCL implementation. This is due to the fact, that the MapOverlap skeleton uses the fast local memory inside its implementation which is hidden from the application developer.

In addition to the performance advantage, the SkelCL program is also significantly simpler than the cumbersome OpenCL implementation. The OpenCL implementation requires 19 lines in total while the SkelCL program only comprises 4. No index calculations or boundary checks are necessary in the SkelCL version whereas they are crucial for a correct implementation in OpenCL.

The OpenCL program in Listing 1.6 is not an optimized, but rather a straightforward version most programmers who are not OpenCL experts would write. Since SkelCL targets such programmers rather than GPU experts, we take this version for comparison with the SkelCL version. An optimized OpenCL version, e.g., using local memory would probably perform better but would definitely require additional low-level code.

4 Conclusion and Related Work

In this paper we showed how the SkelCL library can be extended for developing applications on two-dimensional data. We used an image processing application as a case study. Using SkelCL, such applications can easily benefit from the performance of GPUs. Application developers do not have to be GPU computing experts to achieve good performance, since SkelCL's skeletons exploit the GPU memory hierarchy transparently for the user. The two-dimensional data type significantly simplifies memory management. The SkelCL library is available as open-source software at <http://skelcl.uni-muenster.de>.

Similar approaches have been proposed recently to simplify GPU programming. *SkePU* [3] is a high-level framework for multi-core CPUs and multi-GPU systems offering skeletons similar to SkelCL. A macros-based mechanism allows for using either OpenMP, CUDA or OpenCL as back-end to execute the skeletons, but also restricts the programmer to the back-ends' smallest common set of functions. A skeleton similar to our new proposed MapOverlap skeleton is available in SkePU, but restricted to one-dimensional data. Recently a two-dimensional data type has been added to SkePU, but it is not yet fully integrated in its current version.

In future work we will extend the set of skeletons offered by SkelCL and we will specifically target heterogeneous systems with multiple GPUs.

References

1. NVIDIA CUDA SDK code samples, Version 3.0 (February 2010)
2. NVIDIA CUDA API Reference Manual, Version 4.1 (2012)
3. Enmyren, J., Kessler, C.: SkePU: A multi-backend skeleton programming library for multi-gpu systems. In: Proc. of HLPP 2010 (2010)
4. Kirk, D.B., Hwu, W.W.: Programming Massively Parallel Processors - A Hands-on Approach. Morgan Kaufman (2010)

5. Munshi, A.: The OpenCL Specification, Version 1.2 (2011)
6. NASA. Blue marble: Land surface, shallow water, and shaded topography, <http://visibleearth.nasa.gov/view.php?id=57752>
7. University of Southern California SIPPI Image Database. Girl (lena, or lenna), <http://sipi.usc.edu/database/database.php?volume=misc>
8. Rabhi, F.A., Gorlatch, S. (eds.): Patterns and skeletons for parallel and distributed computing. Springer (2003)
9. Steuwer, M., Kegel, P., Gorlatch, S.: SkelCL – A portable skeleton library for high-level GPU programming. In: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops, IPDPSW, pp. 1176–1182 (2011)

Structured Data Access Annotations for Massively Parallel Computations*

Marco Aldinucci¹, Sonia Campa², Peter Kilpatrick³, and Massimo Torquati²

¹ Computer Science Department, University of Torino, Italy
`aldinuc@di.unito.it`

² Computer Science Department, University of Pisa, Italy
`{campa,torquati}@di.unipi.it`

³ Computer Science Department, Queen's University Belfast, UK
`p.kilpatrick@qub.ac.uk`

Abstract. We describe an approach aimed at addressing the issue of joint exploitation of control (stream) and data parallelism in a skeleton based parallel programming environment, based on annotations and refactoring. Annotations drive efficient implementation of a parallel computation. Refactoring is used to transform the associated skeleton tree into a more efficient, functionally equivalent skeleton tree. In most cases, cost models are used to drive the refactoring process. We show how sample use case applications/kernels may be optimized and discuss preliminary experiments with FastFlow assessing the theoretical results.

Keywords: algorithmic skeletons, parallel design patterns, refactoring, data parallelism, cost models.

1 Introduction

The structured parallel programming approach has abstracted the concept of control and data parallelism by means of *skeletons* [10], which are well known *patterns of control* [8]. Control parallelism is conceived, designed and implemented as a graph of nodes (a skeleton), each node representing a function. A stream of independent tasks flows through the graph: when each node's inputs are available it computes producing output which is sent to its connected nodes. On the other hand, *data parallel skeletons* describe a pattern of computation defining how to access data in parallel and the function which has to be applied to data partitions to get the final result. Traditionally, orthogonality between control parallelism and data parallelism has been dealt with using two-tier models in which control/stream-driven approaches were enhanced with data parallel capabilities, possibly with parallel data structures exposing collective operations [13] and vice versa. However, control parallel and data parallel oriented approaches

* This work has been supported by European Union Framework 7 grant IST-2011-288570 “ParaPhrase: Parallel Patterns for Adaptive Heterogeneous Multicore Systems”.

often lack the ability to describe efficiently applications in which both concerns are exploited because of the intrinsically different means by which parallelism is expressed and, sometimes, optimized. An efficient distribution of tasks in a control driven environment could be invalidated by a poor data access policy, and vice versa [14].

In this paper we sketch a new approach to confronting the control versus data parallel dichotomy based on the idea that: *i)* data and control parallel concerns needs to be independently expressed since they describe orthogonal aspects of parallelism, and *ii)* data accesses and control parallel patterns need to be *coordinated* in order to efficiently support the implementation of parallel applications. While exploiting parallelism through patterns is not a new approach [11] and coordination efforts have been made in the past in terms of languages or frameworks[17,12], the idea proposed in this work is that such coordination can be expressed by annotating the graph of control implicitly defined by the skeletons with information regarding the data access. Moreover, we will show how such annotations can be used to drive optimizations in the implementation and execution of the graph.

2 The Skeleton Framework

The skeleton system considered includes control (i.e. stream) and data parallel skeletons, modelling the more common and general parallelism exploitation patterns. Our skeleton set is defined by the following grammar:

$$\begin{aligned} Skel ::= & Seq(id) | Pipe(Skel, Skel) | Farm(Skel) | \\ & Map(Skel, Splitter, Composer) | Reduce(Skel) \end{aligned}$$

These skeletons represent well-known parallelism exploitation patterns[4]: Seq wraps existing sequential code, Pipe/Farm are stream parallel skeletons processing streams of items and Map/Reduce are data parallel skeletons processing collections of data. In contrast with many skeleton frameworks (including SkeTo [16], Muesli [9] and SkePU) which consider only maps over “collection” input data, we assume the one used in P3L [7] and Skandium [15]: the responsibility for specifying how the subtask items are build out of the input data (set) is left to the application programmer, as is the specification of the re-construction of the result from the collection of partial results. In P3L, the programmer is asked to use the formal parameters of the map as actual parameters of the worker skeleton using “star variables”—a kind of $\forall i$ variable—to establish correspondences between the task and the subtask data items. For example, a matrix multiplication map could be defined as:

```
1 map MM in (float a[N][N], b[N][N]) out (float c[N][N])
2   IP in (a[*i][], b[][*j]) out (c[*i][*j])
3 end map
```

The star variables were logically interpreted as forall loop variables. In this case the calls to the inner product worker skeleton, IP, corresponded to the pseudo-code: $\forall i \in [0, N - 1] \forall j \in [0, N - 1] \text{ call(IP, a[i][], b[][], c[i][j])}$ although the

schedule eventually produced by the P3L compiler may have been completely different from the (sequential) schedule implicit in the nested loops. When dealing with collections and complex and compositional data structures, there are some particular data access patterns that recur which describe how each piece of data is combined into the final result. For example, a block of contiguous data implementing a matrix in a “row major” memory organization, could be accessed by rows or by columns, each row could be coupled with every column, with all the couples becoming targets of computation and the output of each such computation representing a single position in the output matrix. Variants of this kind of pattern include those considering each row/column coupled with a whole matrix or with a sub-block. Another pattern of access is that describing stencil, i.e. a block of cells in a fixed or variable range around each item. Some patterns deal with triangular matrices as, for example, the one accessing diagonals or stencils relative to elements on the diagonal.

In our proposal, control parallelism is described by a composition of control and data parallel skeletons (i.e. patterns of computation) but the corresponding graph is enriched by a set of annotations exposing data access patterns. The different combinations of skeleton type and access patterns can fully describe how computation evolves by guaranteeing the orthogonal management of both data and control parallelism and, at the same time, providing a theoretical platform on which we can built optimization strategies for better exploitation of resources, bandwidth, service time, and other performance measures. In the following section we will provide a language for defining data and control concerns and we will underline how they can be orthogonally described in order to facilitate skeleton rewriting and subsequent skeleton implementation.

3 Annotations/Metadata

Each of the skeletons introduced in Sec. 2 may be enhanced using various kinds of annotation, represented as metadata associated with the skeleton tree. In particular, we use annotations related to the functional (e.g. data access patterns) and non-functional (e.g. performance related) aspects. Such annotations are expressed using the following grammars:

```

ParDegreeAnnot ::= pardegree(int)
CodeAnnot      ::= sourcecode(< string >) | library(< string >)
ArchTypeAnnot ::= GPU|CPU
DataAccessAnnot ::= AccessKind <id> by AccessType
AccessKind       ::= READ|WRITE
Accessstype     ::= ROW|COL|ITEM|BLOCK

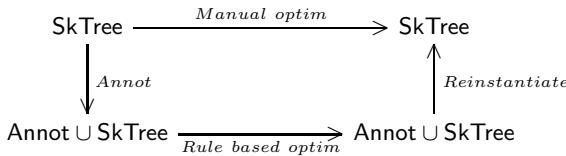
```

The informal semantics associated with the annotations is as follows:

ParDegreeAnnot parallelism degree (if variable)

CodeAnnot associates source and library code with a sequential skeleton.

ArchTypeAnnot target architecture where the skeleton has to be (preferably) executed (heterogeneous CPU/GPU processing element assumed)

**Fig. 1.** Optimization workflow

DataAccessAnnot kind of accesses performed on the input data.

Part of these annotations are provided by the user (e.g. the **CodeAnnot** ones). Others are derived directly from the skeleton source code via a compilation step (e.g. the **DataAccessAnnot** ones). A third class of annotations may be either provided by the application programmer or automatically derived by the compiling tools (e.g. the **ParDegree** ones).

The focus here is on the **DataAccessAnnot** annotations. These are used to optimize the execution of a skeleton program as detailed in Sec. 4 relative to data placement in memory, to communication and synchronization management and also to computation partitioning among the available processing elements. The general idea is summarized in Fig. 1. The skeleton program provided by the application programmer is given to a compiler tool (the “Annot” arrow in the Figure) which produces an annotated skeleton tree. This compiler tool is a kind of abstract interpreter. The annotated skeleton tree is parsed and navigated by the optimizer which eventually produces a different skeleton tree. This new skeleton tree may differ in both annotations (e.g. it has the same shape as the original one but hosts different annotations) and tree shape (e.g. it hosts different, or differently connected, nodes). The optimization phase (the “Rule based optim” arrow at the bottom of Fig. 1) uses different heuristics stored as refactoring rules, possibly identifying, among the possible rewritings, the one giving the best performance figures according to a given skeleton performance model.

As illustrative example, we will show how our abstract syntax and annotation system can be used to write both pure control/stream-parallel applications and data parallel ones, highlighting the syntax usage, the expressive power and how annotation and parameters can support reasoning about skeleton trees. In Section 4 we will go a step further, using annotations in applications exposing *both* stream and data parallel concerns.

Control Parallelism. A pure control parallel skeleton employs stream parallel skeletons only, for example, $\text{Pipe}(\text{Seq}(f), \text{Farm}(\text{Seq}(g), \text{Seq}(h)))$. Fig. 2 shows a possible instantiation of the corresponding syntax tree. The *Seq* skeleton is annotated with a path reference to the code for the sequential function. The *Farm* skeleton is provided with two parameters (the number N of workers and the skeleton implementing each replica) and will be annotated with the actual parallelism degree. *Pipe* is defined in terms of its stages as parameters.

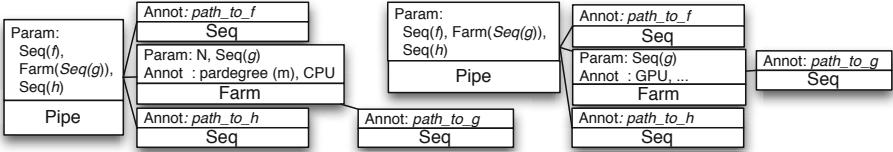


Fig. 2. The syntax tree of $\text{Pipe}(\text{Seq}, \text{Farm}(\text{Seq}))$

In heterogenous architectures (for instance, those provided with CPUs and GPUs) the farm tree could further be annotated with the available alternatives.

Data Parallelism. Matrix multiplication is a traditional example of data parallelism since it can be written as a map whose function is represented by the sequential inner product applied in parallel to each row of the input matrix A coupled with each column of the input matrix B and getting a single item of a matrix C as result¹. From the expression $\text{Map}(\text{Seq}(IP), \underline{\text{in}} A[*i][], B[][*j], \underline{\text{out}} C[i][j])$, the annotated tree in Fig. 3-left can be derived. Note that the expression $A[*i][]$

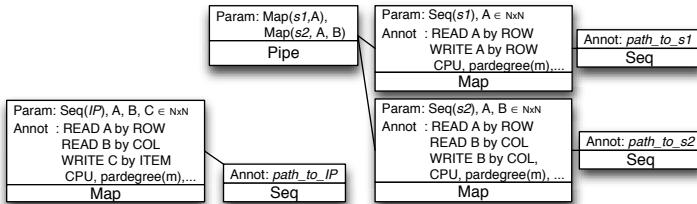


Fig. 3. Left: the syntax tree of $\text{Map}(\text{Seq}(IP))$; Right: a more complex syntax tree

denotes a matrix A accessed by rows and, as a consequence, it defines a *set* of blocks, each represented by one of those rows. For example, considering a matrix A with elements a_{ij} then $A[*i][] = \{[a_{11}, a_{12}, a_{13}], [a_{21}, a_{22}, a_{23}], [a_{31}, a_{32}, a_{33}]\}$. In the case of the expression $A[][*i]$ —denoting access by columns—the set will change accordingly in $A[][*i] = \{[a_{11}, a_{21}, a_{31}], [a_{12}, a_{22}, a_{32}], [a_{13}, a_{23}, a_{33}]\}$. We will employ set operators, where needed. So, for example, we may write that $A' ⊂ A[*i][]$ if and only if A' is a subset of $A[*i]$. Moreover, in the following section, when the structure of such a set is not relevant, we will simply denote it as a result of the *splitting* or *combining* function inside a map definition. Thus, we can write $\text{Map}(f, sp, \underline{\text{out}} A[i][j])$ for some sp , to denote *any* kind of reading access.

¹ In our syntax, *in* and *out* are keywords introducing the map parameters, that is defining the split and compose map policies.

4 From Metadata to Optimization

We now show by examples how the abstract syntax may be used to write applications incorporating both control and data parallelism by coordinating orthogonal concerns and to drive skeleton rewriting in such a way that optimizations are achieved.

Access Driven Optimization. Suppose to have the following abstract description:

$$\text{Pipe}(\text{Map}(\text{Seq}(s1), \text{in } A[*i]\text{[], out } A[*i]\text{[]}), \text{Map}(\text{Seq}(s2), \text{in } B\text{[][*j] } A[*i]\text{[], out } B\text{[][*j]}))$$

for source code $s1$ and $s2$ and $N \times N$ matrices, A and B . The definition allows us to annotate the skeleton tree with the information related to the two maps as depicted in Fig. 3, right.

In the example, a pipe is composed by two maps m1 and m2 . Both access the same dataset \mathbf{A} (a matrix) and apply a function on its rows in sequential stages; m2 takes also the data set \mathbf{B} as input. From [6,2,3] the following rule holds

$$\text{Pipe}(\text{Map}(f_1, sp_1, cm_1), \text{Map}(f_2, sp_2, cm_2)) \equiv \text{Farm}(\text{Map}(\text{Comp}(f_1, f_2), sp', cm'))$$

where the Comp skeleton computes in sequence the two functions. However, from the annotation provided by the data access we can argue that *i)* m2 is functionally dependent on m1 since it accesses matrix \mathbf{A} , written by m1 ; *i)* m2 accesses \mathbf{A} by the same policy used by m1 to write it, i.e. they access the matrix by row and extend the input data space by accessing matrix \mathbf{B} , too. Both conditions can be formally defined by the inclusion $sp_2 \supset cm_1$, since the set of annotations defined by sp_2 in reading mode includes those defined by cm_1 for writing mode. As a consequence, we could merge the two maps in order to save read and write memory accesses and to have a single map *i)* whose computing elements take as input the whole input data (the union of both map input data); *ii)* their computation function is the composite of the previous one, as the rewriting rule suggests. The rewriting process leads to the configuration of a *new* instance of annotated skeleton tree. Such example gives a first idea of a transformational rule that may be expressed as follows

$$\frac{sp_2 \supset cm_1}{\text{Pipe}(\text{Map}(f_1, sp_1, cm_1), \text{Map}(f_2, sp_2, cm_2)) \equiv \text{Farm}(\text{Map}(\text{Comp}(f_1, f_2), sp_1, cm_2))}$$

In other words this rule introduces some access constraints to the applicability of the well-known transformation rule.

Architecture Driven Optimization. As already suggested in [1] the skeleton tree could be annotated also with information related to the target architecture at hand in order to optimize mappings and/or distribution of data. As an example, let us consider a system S provided with n CPUs and r GPUs defined as $S = \{cpu_1, \dots, cpu_n, gpu_1, \dots, gpu_r\}$ and the following skeleton definition, for some source code f . As already seen in Fig. 3, the abstract syntax tree of such definition could be a map whose computing elements are located each on a separate CPU, thus involving a huge amount of data transfer, since the matrix is not

shared and parallelism is exploited in terms of every single item of the matrix. As an alternative to this schema, taking advantage of the GPU subsystem, the skeleton could be rewritten as a map of two maps (i.e. how many the number of GPUs), each computing on a block of data as defined by the language. Thus, the preceding portion of code is represented by

$$\text{Map}_{CPU}(\text{Map}_{GPU}(s, \text{in } A'[*i][], \text{out } A'[*i][], \text{in } A[*r][], \text{out } A[*r][]))$$

where $*^r$ specifies the distribution of r partitions of A 's rows. In other words, Map_{CPU} distributes A by r partitions (blocks) of rows; each partition is taken as input by Map_{GPU} which applies its own policy (distribution by rows) on its block. The syntax tree is represented in Fig. 4, Left which is a map of r Maps

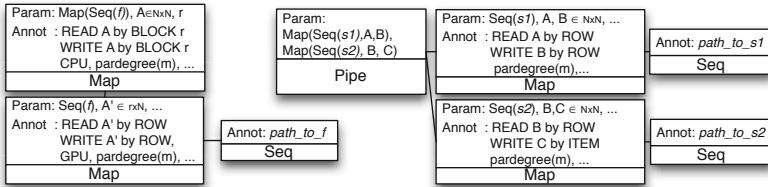


Fig. 4. Map of map on GPU (left); Pipe of two Map (right)

(r being a parameter of Map_{CPU}), each computing a block of data as defined by the language, on a GPU each. Assuming the availability of a cost model $C(\text{Map}_{cpu}(\text{Map}_{gpu}(s)))$ able to qualify the performance of both skeleton trees, our system can be enriched by the following transformation for some skeleton s

$$\begin{aligned} C(\text{Map}_{cpu}(s, sp_1, cm_1)) &> C(\text{Map}_{cpu}(\text{Map}_{gpu}(s, sp_2, cm_2), , sp'_1, cm'_1)) \\ \text{Map}_{cpu}(s, sp_1, cm_1) &\rightarrow \text{Map}_{cpu}(\text{Map}_{gpu}(s, sp_2, cm_2), sp'_1, cm'_1) \end{aligned}$$

asserting that the benefit of moving the execution from CPUs to GPUs depends on provisional costs sustained in the two access policies.

Operator Driven Optimization. With this example we will highlight that our system of rules could be enriched by operators able to manipulate data in order to drive optimizations at an abstract level, thus hiding implementation details.

Let us consider the transposition operator which, given a matrix A , defines a new matrix A^T such that $\forall i, j. A^T[i, j] = A[j, i]$ and $(A^T)^T = A$. Let us suppose that we have the following skeleton definition

$$\text{Pipe}(\text{Map}(s1, \text{in } A[*i][], \text{out } B[*i][]), \text{Map}(s2, \text{in } B[][*j], \text{out } C[i][j]))$$

for some skeleton $s1, s2$ and annotated as depicted in Fig. 4, Right. In this example, we meet a pattern in which one stage writes a matrix and the following stage reads its transposition. A possible optimization involves the first stage placing data into memory and/or computation elements in order to better exploit locality effects. However, this is a valuable strategy only if the block of data on

which the transposition operator has to be applied is small enough to be hosted at cache level; on the contrary, if memory accesses are required to solve cache misses in the reading phase, locality exploitation can raise the cost of referencing data. Both cases can be evaluated by some provisional, qualitative analysis that a cost model monitoring the execution could perform, on the basis of the target architecture and the actual instances of data and skeletons. For simplicity, we will denote using \overline{C} a boolean function evaluating to *true* if such consideration is worthwhile in the context of the actual skeleton instance. Thus, the refactoring rule becomes

$$\frac{P_2 = P_1^T \wedge \overline{C}(P_1^T) \rightarrow \text{true}}{\text{Pipe}(\text{Map}(f_1, sp_1, \text{out } P_1), \text{Map}(f_2, \text{in } P_2, cm_2)) \rightarrow \text{Pipe}(\text{Map}(f_1, sp_1, \text{out } P_1^T), \text{Map}(f_2, \text{in } P_1^T, cm_2))}$$

where P_1 and P_2 represent blocks of data, P_1^T represents the transpose of P_1 and sp_1, cm_2 define generic splitting and combining policies not influencing P_1 .

5 Experimental Results

We describe some experimental results aimed at validating the refactorings discussed in Sec. 4. The results discussed here have been achieved using FastFlow, our experimental skeleton framework targeting multicore architectures [5].

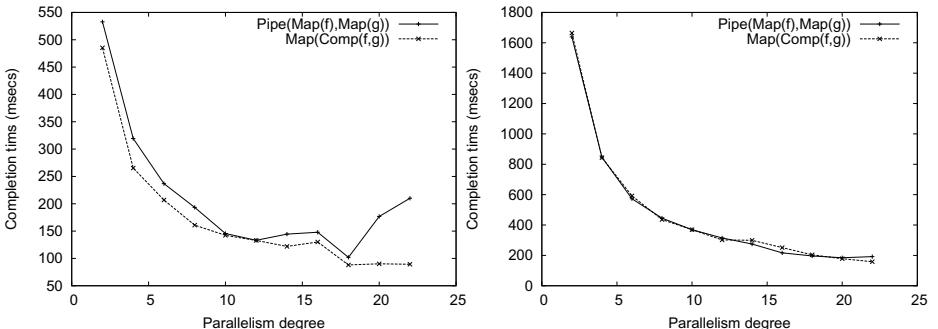


Fig. 5. Effect of map fusion refactoring: small grain (left) and large grain (right)

Access Driven Optimization. We implemented in FastFlow two versions of a program corresponding to the original $\text{Pipe}(\text{Map}(f), \text{Map}(g))$ structuring and to the structuring resulting from fusion, that is $\text{Map}(f, g)$, and we measured the performances on a 24 core Magny Cours (Opteron 6174) architecture². The results are shown in Fig. 5. When the amount of time spent scattering and gathering data to/from the map workers is large enough with respect to the time spent in computing the single map worker (fine grain, left plot) map fusion clearly outperforms the original program. When the time spent in computing

² FastFlow currently does not support a primitive Map skeleton. We used a prototype implementation that will be available with the next FastFlow release.

map workers is considerably larger than the time needed to scatter and gather data to and from the workers, the fused version of the programs performs more or less like the original program. It is worth pointing out that as the parallelism degree increases (and therefore the computation grain decreases), the fused version becomes competitive with respect to the original program version.

Architecture Driven Optimization. The feasibility of refactoring code in such a way that a map originally targeting CPU cores only is transformed into a map targeting CPU cores and GPUs has already been demonstrated in [1]. There we have shown not only that using both CPU cores and GPUs improves the performance of programs with respect to the performances achieved when using only CPU cores, but also that an automatic scheduling procedure may be set up which dynamically uses GPUs and CPU cores to achieve optimal load balancing and, therefore, performances.

Operator Driven Optimization. We took into account the operator driven refactoring discussed at the end of Sec. 4. We implemented a two stage pipeline in FastFlow with both stages computing a function from bi-dimensional float arrays to bi-dimensional float arrays. The functions computed by the stages were very light. We measured the performances of the program run on three different architectures, namely an Intel Xeon Nehalem, an Intel i3 and AMD Magny Cours architecture. We developed two versions of the program: in the first version the first pipeline stage produces a matrix in “WRITE BY ROW” fashion and the second stage reads that matrix “READ BY COLUMN”, while in the second version the first stage produces the transposed matrix and therefore the second stage processes it “READ BY ROW”. The following table summarizes the resulting average computation times, in milliseconds.

	1st (<i>ByRow</i> → <i>ByCol</i>)	2nd (<i>ByRow</i> ^T → <i>ByRow</i>)	% improvement
i3	8786.52	6303.34	28.26
Nehalem	7971.74	5886.94	26.15
Magny Cours	12918.99	11287.23	12.63

The times are for computation of a stream of 100 tasks, each relative to a 1024×1024 floating point matrix. The computation performed on each matrix is negligible—more or less of the “size” of an assignment—for both first and second stage. The refactored version clearly outperforms the original. The smallest performance improvement is on the Magny Cours architecture, which notably sports the smaller memory bandwidth among the three architectures considered here.

6 Conclusions

We outlined a skeleton framework with annotations supporting performance driven refactoring relying on the existence of both suitable skeleton performance

models and the capability to automatically refactor code via rewriting rules. We presented experimental results assessing the approach, which will be used within ParaPhrase WP2 (“Algorithmic skeleton”) activities.

References

1. Aldinucci, M., Danelutto, M., Kilpatrick, P., Torquati, M.: Targeting heterogeneous architectures via macro data flow. *PPL* 22(2) (2012)
2. Aldinucci, M.: Automatic program transformation: The Meta tool for skeleton-based languages. In: Gorlatch, S., Lengauer, C. (eds.) *Constructive Methods for Parallel Programming. Advances in Computation: Theory and Practice*, ch. 5, pp. 59–78. Nova Science Publishers, NY (2002)
3. Aldinucci, M., Danelutto, M.: An operational semantic for skeletons. Technical Report TR-02-13, University of Pisa, Dip. Informatica, Italy (July 2002)
4. Aldinucci, M., Danelutto, M.: Skeleton based parallel programming: functional and parallel semantic in a single shot. *Computer Languages, Systems and Structures* 33(3-4), 179–192 (2007)
5. Aldinucci, M., Danelutto, M., Kilpatrick, P., Meneghin, M., Torquati, M.: Accelerating Code on Multi-cores with FastFlow. In: Jeannot, E., Namyst, R., Roman, J. (eds.) *Euro-Par 2011, Part II. LNCS*, vol. 6853, pp. 170–181. Springer, Heidelberg (2011)
6. Aldinucci, M., Gorlatch, S., Pelagatti, S., Lengauer, C.: Towards parallel programming by transformation: The fan skeleton framework. *Par. Algorithms and Applications* (2001)
7. Bacci, B., Danelutto, M., Orlando, S., Pelagatti, S., Vanneschi, M.: P^3L : A structured high-level parallel language, and its structured support. *Concurrency - Practice and Experience* 7(3), 225–255 (1995)
8. Bromling, S., MacDonald, S., Anvik, J., Schaeffer, J., Szafron, D., Tan, K.: Pattern-based parallel programming. In: Proc. of Int. Conf. on Par. Processing. IEEE Comp. Society, Washington, DC (2002)
9. Ciechanowicz, P., Poldner, M., Kuchen, H.: The muenster skeleton library muesli - a comprehensive overview (07) (2009)
10. Cole, M.: Algorithmic skeletons: structured management of parallel computation. MIT Press, Cambridge (1991)
11. Díaz, M., Rubio, B., Soler, E., Troya, J.M.: Integrating Task and Data Parallelism by Means of Coordination Patterns. In: Müller, F. (ed.) *HIPS 2001. LNCS*, vol. 2026, pp. 16–27. Springer, Heidelberg (2001)
12. Gelernter, D., Carriero, N.: Coordination languages and their significance. *Commun. ACM* 35(2), 97–107 (1992)
13. Kuchen, H.: A Skeleton Library. In: Monien, B., Feldmann, R.L. (eds.) *Euro-Par 2002. LNCS*, vol. 2400, pp. 620–629. Springer, Heidelberg (2002)
14. Kuchen, H., Cole, M.: The Integration of Task and Data Parallel Skeletons. *PPL* 12, 141–155 (2002)
15. Leyton, M., Piquer, J.M.: Skandium: Multi-core programming with algorithmic skeletons. In: Proc. of PDP, pp. 289–296. IEEE Comp. Society (2010)
16. Matsuzaki, K., Iwasaki, H.: A library of constructive skeletons for sequential style of parallel programming. In: InfoScale 2006, p. 13. ACM Press (2006)
17. Rauber, T., Rünger, G.: A coordination language for mixed task and data parallel programs. In: Proc. of the 1999 ACM Symposium on Applied Computing, SAC 1999, pp. 146–155. ACM, New York (1999)

PROPER 2012: Fifth Workshop on Productivity and Performance – Tools for HPC Application Development

Bettina Krammer

Exascale Computing Research
Université de Versailles St-Quentin-en-Yvelines
45 Avenue des Etats-Unis, 78000 Versailles, France
bettina.krammer@uvSQ.fr

Using simulation codes in science and engineering has become commonplace. Writing such code and ensuring that it runs correctly and efficiently also on large numbers of processors and cores is, however, still challenging. Software tools can assist developers of parallel applications in their often tedious tasks of debugging, correctness checking, measuring and analyzing performance. Thus, such tools can help accelerate the development process of complex simulation codes considerably.

The PROPER workshop series seeks contributions on the development of debugging and performance tools and their integration in the software stack comprising the operating system, compilers, runtime environments, libraries, middleware, other tools, and applications. It also encourages tools developers and application developers, likewise, to report about approaches and success stories of how improvements in application performance, scalability, reliability, or productivity can be achieved using tools.

This year's workshop – already the fifth edition of the PROPER series – has received thirteen paper submissions in total, of which six were selected for presentation and publication. Each paper was evaluated by three reviewers.

The workshop was organized in three sessions. The first session featured the invited talk by Georg Hager on *Performance Engineering: From Numbers to Insight*, presenting a process of performance modelling, application monitoring and benchmarking to gain insight into the interaction between hardware and software.

Different instrumentation techniques were presented in the second session with three papers describing their respective approaches.

- *Runtime function instrumentation with EZTrace*, by Charles Aulagnon, Damien Martin-Guillerez, François Rué and François Trahay: this implementation permits to collect data at the entry and the exit of a function.
- *Compiler Help for Binary Manipulation Tools*, by Tugrul Ince and Jeffrey K. Hollingsworth: augmenting assembly files with information about basic blocks and their connections can speed up binary parsing considerably.
- *On the Instrumentation of OpenMP and OmpSs Tasking Constructs*, by Harald Servat, Xavier Teruel, Germán Llort, Alejandro Duran, Judit Giménez, Xavier Martorell, Eduard Ayguadé and Jesús Labarta: instrumenting task-based programs and displaying traces is enabled by close interaction between compiler, parallel runtime, and performance extraction library.

Finally, the third session addressed performance measurement and analysis tools and practices for performance tuning.

- *Strategies for Real-Time Event Reduction*, by Michael Wagner and Wolfgang E. Nagel: this paper explores different approaches towards complete in-memory event tracing.
- *A Scalable InfiniBand Network Topology-Aware Performance Analysis Tool for MPI*, by Hari Subramoni, Jerome Vienne and Dhabaleswar K. (DK) Panda: the INTAP-MPI tool allows users to analyze and visualize the communication pattern of HPC applications on any InfiniBand network.
- *Performance patterns and hardware metrics on modern multicore processors: Best practices for performance engineering*, by Jan Treibig, Georg Hager and Gerhard Wellein: the authors define typical performance patterns and their metrics signatures and demonstrate their generic concepts with real-world usecases.

The workshop proceedings include an abstract of the invited talk and the six contributed papers in order of their presentation. The actual presentations are available for download at the workshop website¹.

As Workshop Chair, I wish to acknowledge all those contributing to the success of this workshop, in particular, the authors of all submitted papers, the members of the Steering and Programme Committees and all the reviewers, the organizers of Euro-Par 2012 and, last not least, the speakers and the attendees of the workshop.

The PROPER workshop series was founded and is being supported by the *Virtual Institute - High Productivity Supercomputing (VI-HPS)*, an initiative to promote the development and use of HPC programming tools. Having received so much positive feedback for this workshop series so far, it is planned to organize the next PROPER workshop again in conjunction with Euro-Par 2013.

¹ <http://www.vi-hps.org/proper2012ws/>

Performance Engineering: From Numbers to Insight

Georg Hager

Erlangen Regional Computing Center (RRZE)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstr. 1, 91058 Erlangen, Germany
georg.hager@fau.de

The ultimate purpose of running simulation tasks on high performance computers is to solve numerical problems. The *performance* of an algorithm, or rather an implementation, is significant in several respects: Either a given problem should be solved in the least possible amount of time or a larger problem should be solved in an “acceptable” time; in both cases, the used resources must be utilized as efficiently as possible so that overall throughput and return on investment are maximized for all users of a system.

Reaching the latter goal implies that the user has some concept of what the “maximum possible performance” of their code is. More often than not, application programmers employ code optimizations without a clear concept of the expected gain. As a result, vast computational resources are wasted. Performance analysis, modeling, and engineering approaches are required to remedy this situation.

An analysis of performance properties naturally starts at the core and chip level, since this is where the actual numerical “work” is done. In-core execution delays, bottlenecks on the chip level, and communication overhead are typical factors that can lead to a deviation from “ideal” performance numbers. A *performance model*, which is typically constructed for every hot spot or loop in a program, predicts the maximum possible performance. The purpose of such models is not so much to predict but to *describe* performance behavior, often with respect to parameters such as problem size or machine properties. When the measured performance numbers deviate from the model, an opportunity arises to learn more about the hardware, the software, or the interaction of both.

The construction of a useful model could be a non-trivial task, however. One of the crucial inputs is a static analysis of the algorithm and/or the code of a hot spot, e.g., in terms of work (flops) performed, dependencies, data transfers, inter-process

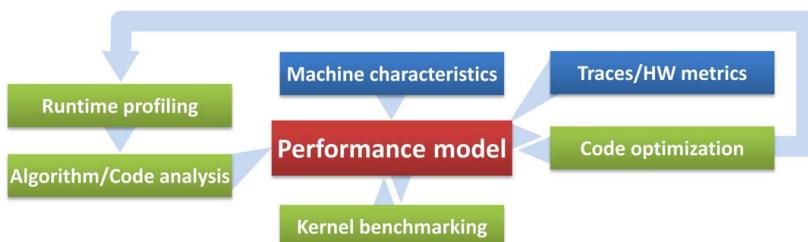


Fig. 1. The structured performance engineering process.

communication, etc. A comparison with documented machine characteristics (maximum instruction throughput/latency, SIMD register width, theoretical latencies and bandwidths of data paths, etc.) can already lead to a first performance estimate. Sometimes features are undocumented, or real code execution cannot saturate a resource. In such cases, *microbenchmarks* can help establish a practical limit; a prominent example is the maximum main memory bandwidth of a chip, which can often not meet the theoretical limit given by the hardware, and should thus be measured by suitable benchmarks such as STREAM. The roofline model [1] and the ECM model [2,3] implement this modeling approach successfully for modern multi- and manycore processors. If power consumption and “energy to solution” is taken into account, a useful model will also estimate the sweet spot of minimum energy to solution with respect to clock frequency, resources used (cores, nodes), and single-thread performance [3].

When it comes to comparing the model with measurements, the pure performance number is just one of many possible data points. *Hardware performance metrics* or *event traces* can be used to further validate the model. For instance, the aggregated count of cache lines transferred between adjacent memory hierarchy levels can be checked against the model prediction. Finally, the insight that has been generated by the performance model and its validation often leads to a clear view on possible optimizations and their potential benefit. Since a code change might incur a substantial modification of the whole application’s timing behavior, a new runtime profile may be in order and the whole workflow starts from the beginning.

The constructed performance model is thus embedded in a *structured performance engineering process* (see Fig. 1). Neither is it possible to automate this completely, nor can it be put into a single “catch-all” formula. One should also note that, although hardware metrics are a central component of performance analysis, they can only provide part of the necessary information. Manual code inspection and the subsequent construction of the performance model is perhaps the most time-consuming part of the process, which also requires substantial experience in some cases.

References

1. Williams, S.W., Waterman, A., Patterson, D.A.: Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Tech. Rep. UCB/EECS-2008-134, EECS Department, University of California, Berkeley (October 2008), <http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-134.html>
2. Treibig, J., Hager, G.: Introducing a Performance Model for Bandwidth-Limited Loop Kernels. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part I. LNCS, vol. 6067, pp. 615–624. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-14390-8_64
3. Hager, G., Treibig, J., Habich, J., Wellein, G.: Exploring performance and power properties of modern multicore chips via simple machine models (submitted), <http://arxiv.org/abs/1208.2908>

Runtime Function Instrumentation with EZTrace*

Charles Aulagnon¹, Damien Martin-Guillerez², François Rué²,
and François Trahay³

¹ ENSEIRB / INRIA Bordeaux Sud-Ouest

charles.aulagnon@gmail.com

² INRIA Bordeaux Sud-Ouest

firstname.lastname@inria.fr

³ Institut Mines-Télécom - Télécom SudParis

francois.trahay@it-sudparis.eu

Abstract. High-performance computing relies more and more on complex hardware: multiple computers, multi-processor computer, multi-core processing unit, multiple general purpose graphical processing units... To efficiently exploit the power of current computing architectures, modern applications rely on a high level of parallelism. To analyze and optimize these applications, tracking the software behavior with minimum impact on the software is necessary to extract time consumption of code sections as well as resource usage (e.g., network messages).

In this paper, we present a method for instrumenting functions in a binary application. This method permits to collect data at the entry and the exit of a function, allowing to analyze the execution of an application. We implemented this mechanism in EZTRACE and the evaluation shows a significant improvement compared to other tools for instrumentation.

1 Introduction

The complexity of modern supercomputer hardware, due to the use of NUMA architecture, hierarchical caches or accelerators, as well as the use of hybrid programming models that mix MPI with OpenMP or PThread make it difficult to exploit efficiently a supercomputer. Optimizing a parallel application requires to understand precisely its behavior, which can be tedious because of the hardware and software stacks.

Generating and analyzing execution traces of an application is a great help for developers who want to optimize their programs. The generation of such traces requires to intercept the calls to a set of key functions – MPI communication primitives, synchronization functions, etc. – and to record events in a file. The instrumentation of a program must not modify its behavior and thus should have an overhead as low as possible.

* We would like to thanks **Julien Pedron** for his work on the packaging of this method in EZTrace – <http://eztrace.gforge.inria.fr>

In this paper, we present a mechanism for intercepting the calls to a set of functions in an application. This mechanism can be used in a performance analysis tool for generating execution trace. We implemented this technique in the EZTRACE framework for performance analysis. The remainder of this paper is organized as follows: Section 2 briefly presents the framework in which our contribution takes place as well as the problems of the instrumentation mechanism that was previously implemented. In Section 3, we present various research related to function instrumentation. The mechanism that we propose and its implementation in EZTRACE are described in Sections 4 and 5. Finally, the results of the evaluation of our implementation are reported in Section 6.

2 The EZTrace Software

EZTRACE [12] is a general framework for generating traces of a program without recompilation. It is able to instrument functions in dynamic libraries and record events in trace files. Several modules are provided that contains function instrumentations for standard libraries like MPI or pthread. For instance, it can save each call (and the timestamp at which the call happened) to MPI_Send into a trace using the mpi module. Moreover, EZTRACE provides a simple mean to generate user-defined modules.

To generate a trace, events have to be recorded before and after each call to specified functions. Recording these events permits to keep track of entry and exit of each function of interest. To do so, EZTRACE uses the LD_PRELOAD mechanism that preloads shared libraries before any other shared libraries. As depicted in Figure 1, the symbols provided by a preloaded library take priority over other symbols when the symbol resolution is done. Since EZTRACE defines a function `f` in a library called `libeztrace-foo` that is preloaded, when the application calls `f`, the function defined by EZTRACE is called. This function records an event, then calls the original function `f` that is defined by the `libfoo` library. When the original `f` ends, EZTRACE records another event and returns to the application.

The LD_PRELOAD mechanism used in EZTRACE is very simple and very efficient. It has enabled the development of a fast and efficient framework for trace generation. However, it can only instrument functions in shared libraries. Thus, EZTRACE cannot intercept calls to functions that are defined in a statically-linked library or within the program.



Fig. 1. Instrumentation of the function `f` with LD_PRELOAD in EZTRACE

3 Related Work

During the optimization of an application, running the program, collecting an execution trace and analyzing it is a great help for identifying the application bottlenecks. Several tools are dedicated to a particular library or programming model like MPI [13], pthread [3] or OpenMP [4]. In order to analyze applications that use several libraries or programming models as well as user-defined functions, generic tools like PABLO [10], TAU [11] or VAMPIRTRACE [8] were developed. These tools provide a set of pre-defined modules for the main programming models used in HPC (MPI, OpenMP, ...) These tools use instrumentation in order to insert probes that record events when a specific function (`MPI_Send` for instance) is called.

The instrumentation can be based on the application source-code: the application is recompiled and functions of interest are instrumented. The main drawback of this technique is that it requires a recompilation of the program.

An alternative solution for instrumentation consists in modifying the binary code for inserting probes. To do so, VALGRIND [9] relies on partial emulation of the machine to enable dynamic rewriting and inserting specific hooks in the system. The list of hooks that Valgrind provides is limited and cannot be easily extended. Moreover, the emulation obviously makes this solution slow in testing high performance programs. PIN [7], DYNAMORIO [1], DYNINST [2] or MAQAO [6] rely on instruction decoding to instrument the code. These tools reverse engineer programs and directly insert *opcodes* anywhere in the binary. This fine-grain instrumentation allows to modify precisely a binary by, for instance, inserting a set of instructions between two *opcodes* or removing some of the *opcodes*. Figure 2 depicts how these tools can be used for instrumenting functions such as `f`: the first *opcodes* of the function are replaced by a trampoline that calls a *prolog* function (which is in charge of recording an event in the output trace) before executing the function *opcodes*. The same operation is performed for inserting a call to the *epilog* function before the end of the function.

These fine-grain instrumentation tools permit to modify precisely an application, but using this kind of mechanism in EZTRACE for coarse-grain tracing

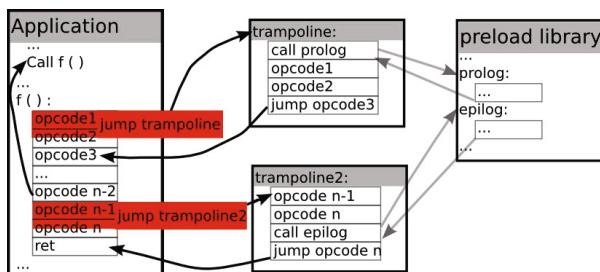


Fig. 2. Instrumentation of the function `f` with DYNINST

(recording an event at the entry and the exit of a function) may cause an overhead. It requires to install several trampolines: one should be installed at the entry of the function, and each exit point should be instrumented. Defining a variable in the `prolog` function and using it in the `epilog` is also complex to implement. Moreover, there is a major issue if the function exit is located at the beginning of a branch: in that case, complex mechanisms have to be used.

4 Coarse-Grain Instrumentation in EZTrace

In order to generate an execution trace of an application, EZTRACE needs to record events at the entry and the exit of a set of functions (MPI primitives for instance). Using `LD_PRELOAD` for that is very convenient as it permits to write a C function that describes how the function should be instrumented. Moreover, it allows to declare local variables that can be used from the entry to the exit of the function. Figure 3 shows an example of function instrumentation in EZTRACE. This code is compiled into a shared library that is preloaded before the application. The `f_orig` variable contains the address of the original function `f` obtained at the initialization of the instrumentation library.

```
double (*f_orig)(int n, double* a);

double f(int n, double* a) {
    record_event(F_ENTRY, n);
    double ret = f_orig(n, a);
    record_event(F_EXIT, ret);
    return ret;
}
```

Fig. 3. Instrumenting the function `f` in EZTRACE

Since this mechanism can only intercept calls to functions located in a shared library, EZTRACE needs an alternative method for statically-linked functions. We propose to design a mechanism similar to `LD_PRELOAD` that is able to instrument these functions. In order to do this, EZTRACE needs to replace the original function `f` with its own version that would be similar to the function depicted in Figure 3. EZTRACE function can record events and use the `f_orig` callback to call the original function `f`. This requires EZTRACE to retrieve the address of the function `f` and assign it to the `f_orig` callback. When using `LD_PRELOAD` with shared libraries, these steps are achieved thanks to the dynamic linker that associates function names to addresses in the memory space. When the function to instrument is not in a shared library, the address of the function is hardcoded in the binary and EZTRACE has to modify the binary program to perform the function interception.

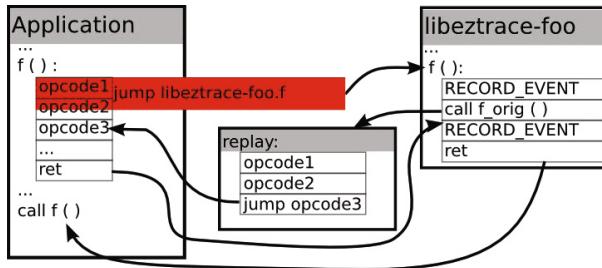


Fig. 4. Instrumentation of the statically-linked function *f* in EZTRACE

The replacement of the function *f* is depicted in Figure 4: a trampoline to EZTRACE’s *f* is inserted at the entry of the original function *f*. The overwritten *opcodes* are relocated in a *replay* section. At the end of the relocated *opcodes*, a jump to the remainder of the function is added. The address of the replay code is assigned to the *f_orig* callback. Thus, when the application calls *f*, the execution flow is redirected to the EZTRACE version of the function. EZTRACE can collect information and record events before using the *f_orig* callback to call the original function *f*. Once the original function *f* returns, the remaining instructions of EZTRACE’s *f* are processed.

This mechanism allows to instrument functions located in a statically-linked program. In case the function to intercept is defined in a shared library, the `LD_PRELOAD` mechanism can be used since the instrumentation code remains similar to the one described in Figure 3. Unlike the methods presented in Section 3, this coarse-grain instrumentation only requires to install a trampoline at the entry of the function to instrument. Thus, it is not necessary to solve the complex problems of multiple exit points in the function or exit points located in a branch. Moreover, the interception function can declare local variables before the entry of the original function to instrument and use them after its exit.

5 Implementation

We have designed and implemented this mechanism in EZTRACE. For instrumenting an application, the following steps are processed:

1. Get the symbols of the function to instrument (*f*), the replacement function (`eztrace_f`), and the callback to the original function (*f_orig*) by reading the binaries.
2. Run the target program tracing with the `ptrace()` system call.
3. Fetch the base address of the library containing `eztrace_f` and *f_orig*.
4. Instrument the function *f*: construct the base trampoline (the jump to `eztrace_f`), the replay code and write the base trampoline and the *f_orig* value in the target process memory.
5. Detach the process and let the target process run.

These steps happen at the application startup, when the process is being launched. In case of an MPI application, these actions are performed by each MPI process. Thus, an overhead due to the instrumentation during the startup phase is to be expected. However, we measured that instrumenting 100 functions with EZTRACE only delays the startup by 280 ms.

5.1 Collecting the Necessary Information

In order to get the relative address of a symbol – such as the function to instrument – EZTRACE uses the *Binary File Descriptor* library (BFD). The collected address is relative to the beginning of the ELF section that contains the function. If the symbol is defined in the program or in a statically-linked library, the base address of the ELF section is known and the absolute address of the symbol can thus be found.

If the symbol is defined in a shared library, the base address of the section is known only once the library is loaded. EZTRACE thus uses the tracing mechanism and wait for the `dlopen()` corresponding to the library (an `open()` on the library followed by a `mmap()`) in order to determine the base address of the library – and thus the address of the symbol – in the address space.

5.2 Function Instrumentation

Once EZTRACE knows the addresses of the required symbols (`f`, `eztrace_f` and `f_orig`), it constructs a base trampoline that jumps from the original function to the instrumentation function when the former is called as depicted in Figure 4. Since installation of the base trampoline overwrites the first instructions of `f`, it is required to relocate them before the installation as shown in Figure 4. In order to determine the size of the overwritten instructions, EZTRACE can rely on the `libopcodes`. If it is not available, EZTRACE implements an alternative method that consists in single-stepping the instructions and observing how the instruction pointer is modified. A memory space is allocated in the target process to install the relocated code (using `mmap()`). Once the base trampoline and the replay code are installed, the address of the replay code is assigned to `f_orig` in the target process (using `ptrace()`).

5.3 Discussion

The model we propose has been implemented in EZTRACE and will be integrated in the 0.8 release as open-source. This model is not specific to EZTRACE and it could be integrated in other performance analysis tools. Our implementation currently only supports Linux, but it is applicable to other systems like Mac OS X or FreeBSD. Porting this method to these systems is part of our future work. Also, in order to construct the trampoline, machine-dependent code is needed. Since we restrict the architecture-specific code to the minimum, this part of the code is very limited (less than 300 lines of code) compared to the architecture-specific code in other tools like DYNINST or PIN. Therefore, porting our code to other architectures requires little effort.

6 Evaluation

Since instrumenting an application and collecting information during its execution increases the number of instructions to execute, it is expected that our implementation implies an overhead compared to running the application without instrumentation. In this Section, we evaluate this overhead and we compare EZTRACE to other tools that can be used for instrumentation : DYNINST 7.1 and PIN 2.11. The results presented here were obtained on an Intel Xeon X5550 at 2.67 GHz.

6.1 Raw Overhead

In order to evaluate the raw overhead of our implementation, we use a program that calls repeatedly an empty function (`compute`) located in a library. The program is linked either statically or dynamically against the library depending on the interception method to analyze. We instrument this program by counting the number of times the program enters the function and the number of times it leaves the function. The instrumentation with PIN and DYNINST thus consists in inserting a call to the counting function (`count_calls`) at the entry and the exit of the function. With EZTRACE, the entry of `compute` is replaced with a call to `ezt_count_calls` that increments a variable, calls `compute` and increments another variable.

Table 1 shows the average duration of an iteration in this program. In the case of the shared library, DYNINST failed to instrument the program. The results show that the instrumentation with PIN, DYNINST and EZTRACE cause a light overhead comprise between 5 and 25 ns.

Since the goal of our instrumentation tool is to generate execution traces, we run the same experiment, but instead of modifying a variable at the entry and exit of `compute`, we record two events in a trace file using the FxT library [5]. The results of this experiment are reported in Table 2. Recording an event being more time consuming than modifying a variable, the measured overheads are higher. The results show that the overhead of EZTRACE is lower than for DYNINST and

Table 1. Function duration when counting the number of calls to a function

Interception method	no instrumentation	PIN	DynInst	EZTrace
Statically linked library	4.7 ns	20.2 ns	28.8 ns	12.3 ns
Shared library	5.2 ns	24.0 ns	-	11.3 ns

Table 2. Function duration when recording events during a function call

Interception method	no instrumentation	PIN	DynInst	EZTrace
Statically linked library	4.7 ns	1287 ns	1294 ns	245.2 ns
Shared library	5.3 ns	1293 ns	-	227.4 ns

PIN. However, because of FxT internals, we observed large variations in this overhead gain depending on the computer used.

6.2 Overhead on an Application

In order to evaluate the overhead of the instrumentation on a real life application, we instrumented an OpenMP application that computes a MD simulation. By recording events at the entry and exit of each function of this application, the resulting trace consists in 7,941,671 events.

Table 3. Execution time of a molecular dynamics simulation using 4 OpenMP threads

Interception method	no instrumentation	PIN	DynInst	EZTrace
Execution time (s)	0.45	3.16	3.28	2.26
Overhead (ns / iteration)	+0	+341	+413	+227

The execution times of the application are reported in Table 3. While PIN and DYNINST cause an overhead of approximately 350 ns per event, instrumenting this application with EZTRACE only degrades the performance by 227 ns per event. This is due to the way EZTRACE instruments the functions: instead of inserting several trampolines, EZTRACE only disrupts the processing flow once.

7 Conclusion

Nowadays, high performance computing (HPC) relies on a high level of hybrid parallelism. To analyze HPC programs, tracing tools with low overhead are needed. Performance analysis tools such as EZTRACE can install function hijacks and trace calls to shared library functions using a LD_PRELOAD mechanism. However, this method cannot apply for statically-linked functions.

In this paper, we described a method that permits to instrument a statically-linked function using a mechanism similar to LD_PRELOAD. Our mechanism improvements to other techniques are: 1/ all the instrumentation is done in the process initialization and have minor impact on the process performance, 2/ a clever design of the instrumentation makes our system easy to use by leveraging C calling mechanism and 3/ the use of `ptrace()` to detect library loading and to allocate memory in the target process makes the amount of machine-depend code really low.

Therefore, our method is much less complex than other methods such as the ones implemented in DYNINST and PIN and has better performance. It is also to be noted that DYNINST is particularly difficult to install and relies on non-standard libraries that may be missing in many systems. Also, PIN is for Intel-based CPU only. Our method is fully integrated into the EZTRACE software and we plan to port this mechanism to other systems (Mac OS X and BSD) as well as other CPUs (ARM).

References

1. Bruening, D., Garnett, T., Amarasinghe, S.: An Infrastructure for Adaptive Dynamic Optimization. In: International Symposium on Code Generation and Optimization, CGO 2003 (2003)
2. Buck, B., Hollingsworth, J.: An API for runtime code patching. International Journal of High Performance Computing Applications 14(4), 317–329 (2000)
3. Bull, S.: NPTL Stabilization Project. In: Linux Symposium, p. 111 (2011)
4. Caubet, J., Gimenez, J., Labarta, J., De Rose, L., Vetter, J.: A Dynamic Tracing Mechanism for Performance Analysis of OpenMP Applications. In: Eigenmann, R., Voss, M.J. (eds.) WOMPAT 2001. LNCS, vol. 2104, pp. 53–67. Springer, Heidelberg (2001)
5. Danjean, V., Namyst, R., Wacrenier, P.-A.: An Efficient Multi-level Trace Toolkit for Multi-threaded Applications. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 166–175. Springer, Heidelberg (2005)
6. Djoudi, L., Barthou, D., Carribault, P., Lemuet, C., Acquaviva, J., Jalby, W., et al.: Maqao: Modular assembler quality analyzer and optimizer for itanium 2. In: The 4th Workshop on EPIC Architectures and Compiler Technology, San Jose (2005)
7. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In: PLDI 2005 (2005)
8. Muller, M., Knupfer, A., Jurenz, M., Lieber, M., Brunst, H., Mix, H., Nagel, W.: Developing scalable applications with Vampir, VampirServer and VampirTrace. In: Proceedings of the Minisymposium on Scalability and Usability of HPC Programming Tools at PARCO (2007)
9. Nethercote, N., Seward, J.: Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. In: ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, PLDI 2007 (2007)
10. Reed, D., Roth, P., Aydt, R., Shields, K., Tavera, L., Noe, R., Schwartz, B.: Scalable performance analysis: The Pablo performance analysis environment. In: Proceedings of the Scalable Parallel Libraries Conference, pp. 104–113. IEEE (2002)
11. Shende, S., Malony, A.: The TAU parallel performance system. International Journal of High Performance Computing Applications 20(2), 287 (2006)
12. Trahay, F., Rue, F., Namyst, R., Faverge, M.: EZTrace: a generic framework for performance analysis. In: CCGrid 2011 (2011)
13. Vetter, J., de Supinski, B.: Dynamic software testing of MPI applications with Umpire. In: ACM/IEEE 2000 Conference on Supercomputing, p. 51. IEEE (2006)

Compiler Help for Binary Manipulation Tools

Tugrul Ince and Jeffrey K. Hollingsworth

Computer Science Department University of Maryland College Park, MD 20742
`{tugrul,hollings}@cs.umd.edu`

Abstract. Parsing machine code is the first step for most analyses performed on binary files. These analyses build control flow graphs (CFGs). In this work we propose a compilation mechanism that augments binary files with information about where each basic block is located and how they are connected to each other. This information makes it unnecessary to analyze most instructions in a binary during the initial CFG build process. As a result, these binary analysis tools experience dramatically increased parsing speeds - 3.8x on average.

1 Introduction

Binary analysis is a common operation for performance modeling [16], computer security [18], maintenance [5], and binary optimization [11]. Each of these tasks requires parsing the executable file to identify functions, data segments, and their interaction with each other. However, parsing executables is not a straightforward task and it is painfully slow since it usually requires decoding every single instruction in the binary. At the higher level, even distinguishing code and data is difficult since they are often stored in adjacent memory. All the information about functions and data locations is actually known during various stages of compilation. However, only some of this information is stored in the binary in the form of symbols. Binary analysis tools that operate on these executables have to regenerate the information that is thrown away by the compiler.

In this work, we propose a novel compilation mechanism that stores useful information about the layout of executable files in tables inside executable files. These tables enable identification of basic blocks and provide support for reconstruction of edges between them. Binary analysis tools, including those that aid development of high performance applications, can parse executables faster and more reliably using these tables. We measured a speed-up in parsing up to 4.4x with an average speed-up of 3.8x. Since these tables are stored in a section that is not loaded into the memory during execution, the memory footprint of executables do not change. Running times of these executables also remain unchanged since we do not in any way modify the execution. The overhead in the compilation time and the increase in file size is manageable - both at around 23%.

2 Difficulties of Binary Parsing

The process of analyzing bytes from a file and generating abstractions like instructions, functions and CFGs is called binary parsing. It is a tedious task with many challenges, such as distinguishing code from data. Since both code and data are stored the same way, there is really no easy way of identifying whether a sequence of bytes correspond to code or data. Current parsing techniques use hints to identify code and mark the remaining bytes as data. These hints usually come in the form of symbols representing functions. From symbols, tools either follow a sweeping or a recursive strategy [20]. In the sweeping strategy, tools first use symbols to mark an initial set of functions, then sweep the remaining bytes from the start of the file and mark sequences of bytes that resemble code as program code. In the recursive strategy, tools also start by using symbols to mark the initial set of functions, then locate other code sections following call edges and marking call targets as function entry points, hence program code. In some cases, uncharted regions in the binary are then plugged into machine learning algorithms to identify even more functions and code regions [19].

Functions are composed of one or more, usually several, *basic blocks*. A basic block is a sequence of instructions that contains no control flow instructions except as the last instruction of the block. If the first instruction in a basic block executes, it is guaranteed that all following instructions will execute. It is an abstraction that is used by many types of analyses. Once the functions are identified, their *Control Flow Graphs* (CFGs) are built. A sample CFG can be seen in Fig. 1. Building such a CFG correctly depends on correct identification of basic blocks and the edges between these basic blocks. Therefore, it requires the analysis tool to inspect each instruction in a function. This operation is error-prone, especially on variable-length instruction set architectures where an error in decoding an instruction propagates downstream and make decoding the following instructions harder, or even impossible.

3 Compiler Help

During the build process, compilers construct an internal representation of the source code and generate machine code using this internal representation. However, as soon as the executable file is generated, this internal representation is thrown away. In this paper, we investigate the effects of storing some of this information about the program inside the executable.

3.1 Basic Block and Edge Tables

To speed up building basic block abstractions inside binary analysis tools, we developed a compilation framework that stores the start address and the address of the last instruction of each basic block inside the executable in what we call a *Basic Block Table*.

CFGs require identification of edges between basic blocks. Our system stores the source basic block, the target basic block, and the edge type of each edge

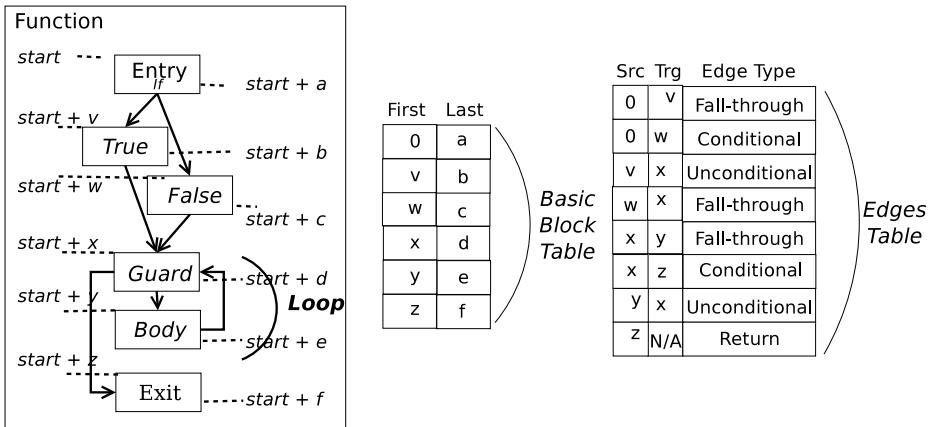


Fig. 1. A sample CFG and associated Basic Block and Edge Tables of a function with entry basic block, an If/Else structure, a loop, and an exit block

in the *Edge Table*. In this table, a basic block is identified by its start address. In some cases, target basic block in an edge can only be known during program execution. Such cases occur when the value of a function pointer or any other indirect branch target¹ depends on the user input. In these cases, we leave the target basic block field blank during generation of the Basic Block Table. Such edges can be filled in by the parser after the program has launched and when the value of the target address can be computed.

Figure 1 illustrates what information is stored in the Basic Block and Edge Tables based on the CFG shown on the left. For this example, assume a, b, c, \dots are the offsets of the last instructions of their corresponding basic blocks from the start of the function. Since there are 6 basic blocks in the CFG, the Basic Block table has 6 rows. The first row represents the first basic block: it starts at offset 0, and its last instruction is located at offset a ; the next row represents the second basic block, and so on. The Basic Block Table is followed by the Edge Table which has 8 rows, one for each edge between basic blocks. The edge from the entry block to the *else* block is represented with the triplet of $<0, w, \text{Conditional}>$ since it is accessed by taking a conditional branch. The edge from the entry block to the *if* block is traversed when the conditional branch is not taken and when execution simply falls-through² to the *if* block. The last row in the figure represents the edge originating from the return instruction. Since the target of a return instruction cannot be determined statically, that field is left blank (marked with N/A in our example).

¹ Although indexed jump tables also use indirect branches, targets of such indirect branch instructions can usually be identified using heuristics.

² ‘Fall-through’ is a type of control transfer where the execution of an instruction is followed by the execution of the next instruction in executable’s address space.

3.2 Compilation Process

Our compilation mechanism mimics a standard compilation process. We developed a tool that modifies assembly files and generates Basic Block and Edge Tables.

One issue in the implementation is the need to support position independent code such as libraries that can be loaded at different addresses in different executions of a program. Using absolute addressing does not work for this type of code. To handle this, the addresses in these tables are stored as offsets from the start of the function.

Another issue arises when some function definitions are merged by the linker. During the build process files that are linked with the *include* directive from a source file are compiled along with the actual source file. As a result all function definitions included from a header file are compiled into the resulting object file. If the same function definition is linked from multiple source files, these function definitions appear in multiple object files that result from the compilation of these source files. To avoid linking problems, these functions are marked as *weak*. Linkers allow only one copy of these weak functions to appear in the final executable - the remaining ones are dropped. Although these functions are identical at the source code level, since compilers perform optimizations individually on each copy of the function, the resulting machine code may be different. As a result, Basic Block and Edge Tables for these functions may differ slightly. Unlike weak functions, tables with the same name cannot be merged. Therefore, our compilation mechanism has to distinguish tables related to functions that have the same name. As a result, we generate table names using a combination of the function name and the name of the file that contains that function. At the end, each table has a different name, and there is a one-to-one mapping between functions and tables.

4 Evaluation

For evaluation, we used a simple binary modification tool we built based on Dyninst [4], an instrumentation and binary analysis tool for HPC applications. Our tool reads in a binary file and rewrites it to disk with simple instrumentation code for basic block counting. Since our analysis deals with every single basic block in the executable, the executable file has to be parsed from top to bottom, correctly locating every basic block. We compared the parsing speed of our tool with that of unmodified Dyninst. Although Dyninst is traditionally known for its dynamic analysis capabilities, it also serves as a static analysis tool with support for basic binary analysis and CFG generation along with binary rewriting. Like other static analysis tools, it makes use of symbols stored in the binaries to improve its analysis, although the existence of symbols is not required in many cases. Therefore, we expect our technique can improve similar static analysis tools in the same fashion.

We first evaluated our system on SPEC CINT2006 [9,15]. SPEC CINT2006 contains a series of CPU-intensive executables that are selected to evaluate the

processor(s) and the memory system. All together, SPEC CINT2006 has about 1,047,000 lines of code.

Our next benchmarks were the PETSc libraries [2]. PETSc (Portable, Extensible Toolkit for Scientific Computation) is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It uses MPI for parallelization. It has linear and non-linear equation solvers and supports C, C++, Fortran and Python. The PETSc suite is composed of about 872,000 lines of code.

Finally we evaluated our system on the popular web browser Firefox (version 9.0.1) and all the shared libraries that ship with the Firefox source code. We evaluated our system on Firefox because its executables are numerous and are relatively large. Moreover, it contains hand-written assembly files and the build process involves using many uncommon compiler options. Therefore, building Firefox has been a valuable test for the robustness of our compilation mechanism. The Firefox suite contains approximately 5,335,000 lines of code.

4.1 Environment

All experiments were carried out on 64-bit x86 machines that run the Linux operating system. SPEC CINT2006 and PETSc benchmarks were tested on a system that has 4 Intel Xeon processors with 6-cores each and 48GB main memory. All our executables except PETSc were serial applications. Therefore, we ran most of our experiments serially on a single core. We used gcc 4.1.2 for building reference executables and as a back-end to our compilation mechanism. Firefox experiments were run on a separate machine due to the idiosyncratic requirements of the Firefox build environment. As a result, Firefox runs were taken on a dual-core machine with an AMD Turion processor at 1.8 GHz with 2GB main memory. On this system, we used gcc 4.6.1 for building reference executables and as our back-end. Since we never compare results across these machines directly, the results are not affected due to using two separate machines.

4.2 Experimental Results

Our first experiment was designed to calculate the time it takes to parse a specific executable using our analysis tool to show how much our tool improves parsing speed. We then ran other experiments to evaluate properties of executables built using our compilation mechanism and identify any trade-offs. Similarly, we compared file sizes after the compilation process. At the end, we tested the runtime performance of these executables in terms of time and memory usage. We ran each timed experiment 5 times and computed the mean. We then normalized our findings with respect to the executables compiled with the gnu compiler suite.

In our experiments, we observed that using basic block and edge tables reduced the parsing time between 58% and 77%, and on average by 73%. Although the file sizes increase by 23% on average, we believe this situation is not prohibitive since the basic block and edge tables are not loaded into memory during the execution of these binary files. We also observed about 23% increase in the

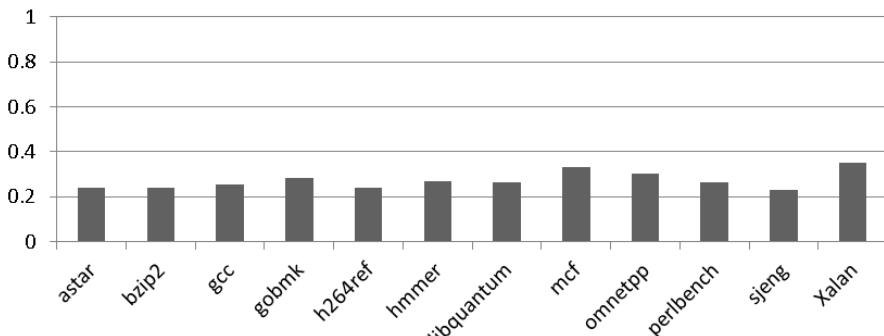


Fig. 2. SPEC CINT2006 Benchmarks: Normalized Parsing Times by Executable

compilation time. Since this is only a one time cost that appears while building executables, and it can be improved drastically by integrating the creation of basic block and edge tables into the compiler rather than a separate assembly pass, we believe this increase is acceptable.

Experimental Parsing Results. Figure 2 shows the normalized parsing times with SPEC CINT2006 benchmarks with respect to regular parsing. We observed a high percentage of speed-up across the board while the average binary parse speed-up is 3.7x (73% improvement over original parsing time).

Our next tests were carried out on executables in *snes* package of PETSc suite. One interesting characteristic of PETSc executables is that PETSc libraries are statically linked into the executable files by default. As a result, each executable contains all functions in PETSc libraries. One can argue that the total cost of parsing these statically-linked executables is higher than the cost if these executables were linked with shared libraries. However, with the help of our compilation mechanism, we reduced the parsing time 76% on average (4.2x speed-up). Due to static linking of all these PETSc libraries in every executable, parsing time is more or less flat across all executables in this set because our tool parses mostly the same set of functions for each executable.

As a final set of executables, we decided to use the Firefox executable and all shared libraries that ship with Firefox. For this set of runs, we operated on those executables that reside in memory when the Firefox web browser is launched. We see a major improvement in parsing time once more as expected. The average drop in the parsing time is 71% (3.5x speed-up) with the worst case reduction of 58%.

As the previous results show, our system considerably increases the parsing speed. Now we want to discuss other evaluation metrics such as file size, compilation time, and memory footprint of executables.

Build Time Metrics. Table 1 gives an overview of our experimental results regarding build time and runtime metrics. In this section, we will discuss the build time metrics: file size and compilation time.

Table 1. Various Properties of our System (All numbers are normalized)

Benchmark Set	File Size Growth		Compilation Time	Running Time	Memory Footprint
	vs. Standard	vs. Debug			
SPEC CINT2006	2.21	1.38	1.25	0.97	1.00
PETSc	1.50	1.09	1.32	0.95	1.00
Firefox	1.17	1.21	1.13	0.94	1.00
Overall Average	1.63	1.23	1.23	0.95	1.00

Since we are adding extra data to executable files, the size on disk unavoidably increases. On average, we are adding about 20 bytes of data to the executable for each basic block, and two extra symbols to the symbol table for each function. Table 1 shows the normalized file sizes across three sets of benchmarks along with the overall average. *Standard* shows the comparison of file sizes when they are built with no debug flag on while *Debug* shows the comparison with the debug flag (*-g*) on. We show both numbers since we realize executables are often built with debug flag on to improve debugging and other binary analyses on these files. The highest increase was observed by the SPEC CINT2006 benchmarks with 121% increase in the average file size with no debug flag on. On average, we observed an increase of 63% with no debug flag on, and 23% with the debug flag on. We assert that this increase is manageable since it does not impact the memory used during execution.

Another evaluation metric we used is the compilation time. Since our compilation mechanism uses an intermediate step to process the assembly code generated by the gcc, our compilations take more time than the original compilations. The bulk of the increase in compilation time comes from the cost of processing an assembly file as text, and writing out a modified assembly file, again as text. Currently, this step is costly and our experiments showed a 23% increase in the compilation time (Table 1). We believe our system can easily be integrated into a mainstream compiler such as gcc. Since compilers already maintain the information we generate using our mechanism, the added cost would be minimal - expected to be the same as the cost of writing those tables to the binary. This improvement remains as future work.

Running Time Metrics. We next looked at the running times of the executables built using our compilation mechanism and their memory footprint. The results are presented in Table 1.

In our experiments we have not experienced any measurable increase in the execution time of the benchmarks. The slight improvement we observed in the running time after using our compilation mechanism is well within the noise of the experiment.

To measure memory footprint change, we evaluated each executable under Valgrind's *massif* tool [17] and measured heap memory and stack memory usage with *--pages-as-heap=yes* flag. Results indicate that both stack and heap memory used by the programs remain about the same, as shown in the final column in Table 1.

5 Related Work

Parsing binary code has been studied extensively in the past. Several researchers created higher level representations of machine code following binary parsing and the disassembly of code. Examples of this approach include Cifuentes and Gough with their decompiler, *dcc* [5], and Emmerik and Waddington with their Boomerang-based decompiler [7]. More recent work concentrates on disassembly of obfuscated code to identify malicious software [12]. Some researchers, such as Aaraj et al. [1], combine static disassembly techniques with dynamic analysis to cope with malware. Similarly, Bruschi et al. attempt to identify malware by building a CFG from binary code and comparing it with those of the known malware [3]. Disassembly techniques also made their way into the mainstream applications: Many common tools such as *gdb*, *objdump*, and *IDA* [10] generate disassembly of binary files. Many researchers build CFGs once the executable file is disassembled. De Sutter's [6] and Theiling's [23] control flow generation algorithms are such examples.

All these systems, including Dyninst [4], make use of the debugging symbols whenever possible. Many tools also perform a best-effort approach to identify function locations if the symbols are not present, such as Harris and Miller's tool [8].

With this work, we let binary analysis tools benefit from the knowledge compilers gather about executables during the build phase. There are several binary analysis tools that can make use of our system. Examples include ATOM [22], EEL [13], Pin [14], Valgrind [17], and Vulcan [21]. All these tools create some sort of internal representation of the binary. Therefore, they all can benefit from using the data stored in Basic Block and Edge tables as much as Dyninst does.

6 Discussion

Parsing executable files is the first step for any CFG-based binary analysis. Our experimental results show that our mechanism clearly speeds up parsing executable files. It is not hard to imagine bundling more information with the binary to speed up other binary analyses, or improve their precision, such as liveness analysis of registers or dependency analysis.

However, there are also shortcomings of our work. One such shortcoming is that our system adds $2n$ more symbols into the symbol table where n is the number of functions. Since symbol tables are highly-optimized, this issue is more of a nuisance than a technical problem. Increased compilation times might also be annoying for large frameworks such as Firefox. However, we expect that integrating our system with a full compiler will substantially speed up compilation.

One improvement to our plain text table based system would be compressing Basic Block and Edge Tables to reduce disk space demand. In our preliminary experiments, we observed that compressing Basic Block and Edge Tables reduced the size of these tables by about 78%. However, since binary analysis tools cannot

read compressed tables directly, they would need to decompress them before first use. We plan to investigate effects of using compressed tables in terms of parsing performance and disk space in our future work.

7 Conclusion

Parsing binary code is the first step for most binary analyses. However, it is costly and imprecise especially on variable-length instruction set architectures. In this work we introduced a novel compilation mechanism that improves the parsing speed of binary files when they are examined by binary analysis tools. Our compiler creates intermediate assembly files, augments them with information about basic blocks and edges between them, and generates executable files using this augmented assembly code.

We implemented an instrumentation program for basic block counting that rewrites a binary to the disk with the instrumentation code using the Dyninst library. We showed that running this analysis code on various benchmarks resulted in up to 4.4x speed-up in parsing time, with an average of 3.8x. Although the size of the binary files increase with extra data in the tables we generate, since these tables are not loaded into memory during execution, the size of the runtime memory image of the executable remains the same as before. Moreover, there is no runtime performance degradation due to these tables.

References

1. Aaraj, N., Raghunathan, A., Jha, N.K.: Dynamic Binary Instrumentation-Based Framework for Malware Defense. In: Zamboni, D. (ed.) DIMVA 2008. LNCS, vol. 5137, pp. 64–87. Springer, Heidelberg (2008)
2. 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>
3. Bruschi, D., Martignoni, L., Monga, M.: Detecting Self-mutating Malware Using Control-Flow Graph Matching. In: Büschkes, R., Laskov, P. (eds.) DIMVA 2006. LNCS, vol. 4064, pp. 129–143. Springer, Heidelberg (2006)
4. Buck, B., Hollingsworth, J.K.: An api for runtime code patching. Int. J. High Perform. Comput. Appl. 14, 317–329 (2000)
5. Cifuentes, C., Gough, K.J.: Decompilation of binary programs. Software: Practice and Experience 25(7), 811–829 (1995)
6. De Sutter, B., De Bus, B., De Bosschere, K., Keyngnaert, P., Demoen, B.: On the static analysis of indirect control transfers in binaries. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2000, Las Vegas, Nevada, USA, June 24–29, pp. 1013–1019 (2000)
7. Emmerik, M., Waddington, T.: Using a decompiler for real-world source recovery. In: Proceedings of the 11th Working Conference on Reverse Engineering, pp. 27–36 (November 2004)
8. Harris, L.C., Miller, B.P.: Practical analysis of stripped binary code. SIGARCH Comput. Archit. News 33, 63–68 (2005)

9. Henning, J.L.: Guest editor's introduction. *SIGARCH Comput. Archit. News* 35(1), 63–64 (2007)
10. IDA: About, <http://www.hex-rays.com/products/ida/> (retrieved March 21, 2012)
11. Ince, T., Hollingsworth, J.K.: Profile-Driven Selective Program Loading. In: D'Ambru, P., Guaracino, M., Talia, D. (eds.) Euro-Par 2010, Part I. LNCS, vol. 6271, pp. 62–73. Springer, Heidelberg (2010)
12. Kruegel, C., Robertson, W., Valeur, F., Vigna, G.: Static disassembly of obfuscated binaries. In: Proceedings of the 13th Conference on USENIX Security Symposium, SSYM 2004, vol. 13, p. 18. USENIX Association, Berkeley (2004)
13. Larus, J.R., Schnarr, E.: EEL: machine-independent executable editing. In: PLDI 1995: Proceedings of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation, pp. 291–300. ACM, New York (1995)
14. Luk, C., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: PLDI 2005: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 190–200. ACM, New York (2005)
15. McGhan, H.: SPEC CPU2006 benchmark suite. Microprocessor Report (2006)
16. Miller, B., Callaghan, M., Cargille, J., Hollingsworth, J., Irvin, R., Karavanic, K., Kunchithapadam, K., Newhall, T.: The Paradyn parallel performance measurement tool. *Computer* 28(11), 37–46 (1995)
17. Nethercote, N., Seward, J.: Valgrind: A program supervision framework. In: Third Workshop on Runtime Verification, RV 2003 (2003)
18. Prasad, M., Chiueh, T.: A binary rewriting defense against stack based overflow attacks. In: Proceedings of the USENIX Annual Technical Conference, pp. 211–224 (2003)
19. Rosenblum, N., Zhu, X., Miller, B., Hunt, K.: Learning to analyze binary computer code. In: Proceedings of the 23rd National Conference on Artificial Intelligence, AAAI 2008, vol. 2, pp. 798–804. AAAI Press (2008)
20. Schwarz, B., Debray, S., Andrews, G.: Disassembly of executable code revisited. In: Proceedings of the Ninth Working Conference on Reverse Engineering, WCRE 2002, pp. 45–54. IEEE Computer Society, Washington, DC (2002)
21. Srivastava, A., Edwards, A., Vo, H.: Vulcan binary transformation in a distributed environment. Technical report, Microsoft Research (2001)
22. Srivastava, A., Eustace, A.: ATOM: a system for building customized program analysis tools. In: PLDI 1994: Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation, pp. 196–205. ACM, New York (1994)
23. Theiling, H.: Extracting safe and precise control flow from binaries. In: Proceedings of the Seventh International Conference on Real-Time Systems and Applications, RTCSA 2000, p. 23. IEEE Computer Society, Washington, DC (2000)

On the Instrumentation of OpenMP and OmpSs Tasking Constructs

Harald Servat^{1,2}, Xavier Teruel¹, Germán Llort^{1,2}, Alejandro Duran^{1,3},
Judit Giménez^{1,2}, Xavier Martorell^{1,2}, Eduard Ayguadé^{1,2},
and Jesús Labarta^{1,2}

¹ Barcelona Supercomputing Center

² Universitat Politècnica de Catalunya

³ Intel Corporation

c/Jordi Girona, 31 08034 - Barcelona, Catalunya, Spain

Abstract. Parallelism has become more and more commonplace with the advent of the multicore processors. Although different parallel programming models have arisen to exploit the computing capabilities of such processors, developing applications that take benefit of these processors may not be easy. And what is worse, the performance achieved by the parallel version of the application may not be what the developer expected, as a result of a dubious utilization of the resources offered by the processor.

We present in this paper a fruitful synergy of a shared memory parallel compiler and runtime, and a performance extraction library. The objective of this work is not only to reduce the performance analysis life-cycle when doing the parallelization of an application, but also to extend the analysis experience of the parallel application by incorporating data that is only known in the compiler and runtime side. Additionally we present performance results obtained with the execution of instrumented application and evaluate the overhead of the instrumentation.

1 Introduction

Current supercomputers offer large computational power by connecting multiple computing nodes using a fast interconnection network. According to the TOP500 list [5], most of the supercomputers use multicore processors which increase the inter-node parallelism. It is not surprising to observe that parallelism has currently hit the desktop segment. However, all the compute power offered by these processors needs some guidance to achieve real parallel execution. And not only this, but guaranteeing that the parallel execution is taking real benefit of the system resources is also an important task.

OpenMP* [7] is a widely known shared memory parallel programming model that allows implementing parallel applications by using a set of compiler directives. The OpenMP runtime deals with the parallel thread execution (including fork, execution and join of the slave threads) and offers the user incremental parallel development by adding the compiler directives gradually. Other parallel programming models may require the developer a major adaptation of the

application (for instance, MPI* [14]) or deal with the thread management (if using the pthread library). In addition, OpenMP allows to express irregular parallelism through a set of new constructs: the OpenMP Tasking constructs (which includes `task` and `taskwait` since version 3.0 and `taskyield` since version 3.1). A task is a unit of parallel work used to express unstructured parallelism that will be executed by one of the threads at a time, but different parts of a task may be executed by different threads.

OmpSs [6] is a parallel programming model based on the OpenMP standard that extends the OpenMP Tasking constructs to support asynchronous task parallelism and execution on heterogeneous devices. Asynchronous parallelism is enabled by the use of data-dependencies between the different tasks of the program (by extending the OpenMP task construct with `input`, `output` and `inout` clauses with respect to the variables used in the task). These data-dependencies are annotated in a task dependency graph created at runtime. The OmpSs runtime starts scheduling tasks that have their dependencies satisfied (i.e., their required data is ready). Then, as soon as tasks finish, the dependency graph gets updated allowing the dependent tasks to become ready for execution.

Performance tools are pieces of software devoted to analyze the performance of applications by looking for bottlenecks. Among the variety of performance tools, Paraver [22] is a flexible parallel program visualization and performance analysis tool. Its key features include: detailed quantitative analysis of program performance, concurrent comparative analysis of different traces, support for mixed message passing and shared memory applications, and customized semantics of the visualized information.

In this paper, we present the result of the synergy between OmpSs runtime and a Paraver trace generation tool. We not only demonstrate the utility of being capable of emitting the internal values of the OmpSs runtime into tracefiles, but we also improve the performance analysis life-cycle by providing an integrated mechanism to gather the application performance data. We present a new display mechanism for tasking constructs in the visualization tool that facilitates the understanding of the execution of the application from the task perspective. We also evaluate the overhead of our proposal by using a set of benchmarks. And, finally, we demonstrate the usefulness of this work by analyzing an application.

The remainder of this paper is organized as follows: we present a summary of the tools involved in this document in Section 2. Section 3 follows, where we describe our implementation. Section 4 describes the experimental results using several benchmarks. We discuss the related work in Section 5. Finally, we draw some conclusions and present future directions of this work in Section 6.

2 Background

Exrae [1] is a performance extraction tool that generates Paraver [22] tracefiles. Exrae requires a two-stage execution process. The first step gathers data like hardware performance counters and source code references from multiple parallel runtimes (including OpenMP, MPI, CUDA* and pthread) by using either instrumentation or sampling on compiled optimized binaries as long as the

application executes. The second step generates the tracefile and occurs after the application execution. The information gathered by Extrae depends on the parallel runtime and it is only limited by the available documentation.

A Paraver tracefile is a container of timestamped records related to the actual execution of an application. The performance information is mapped into the application resources that were assigned in the execution up to three levels (namely: application, process and thread). Among the types of records that can be stored in a tracefile, we focus on two of them: events and relationships. Events are punctual information that occurs in an object (thread or process) of the application. Typical events include entry and exit points of user routines, hardware counter values, callstack information, among others. On the other side, relationships refer to MPI point-to-point communications between two partners of an application. Information contained in a relationship includes: the two objects involved, the timestamps for the point-to-point operations, the size and tag of the communication.

The OmpSs programming model is built on top of the Mercurium compiler [2] and the Nanos++ runtime system [3]. On the one hand, Mercurium is a source-to-source compilation infrastructure aimed at fast prototyping. It is mainly used to implement OpenMP and OmpSs, but since it is extensible, it has been used to implement other programming models and compiler transformations. Extending Mercurium is achieved by using a plugin architecture, where plugins are dynamically loaded by the compiler according to the given user configuration. Code transformations are implemented in terms of source code (that is, there is neither a need to modify nor a need to know the internal syntactic representation of the compiler). On the other hand, Nanos++ is an extensible runtime library designed to serve as a runtime support in parallel environments. Nanos++ supports OmpSs, among other programming models, and it is responsible for scheduling and executing parallel tasks specified by the compiler based on the constraints specified by the user.

3 Implementation

We discuss in this section the modifications applied to the Nanos++ runtime library and to the Extrae instrumentation package. The modifications discussed below include how to generate instrumentation events, how to manage task switching from the Nanos++ runtime point of view and how to postpone such management to the Extrae post-process phase. We show also the modifications applied when visualizing the tasks without having to modify the display tool.

3.1 Implementation of the Instrumentation Plugin for Nanos++

The Extrae instrumentation library provides an API to emit events into the tracefile. Despite being accessible from Nanos++, Extrae needs a mechanism to identify which thread is executing the function called. We have added a call to the API of Extrae that allows identifying the calling thread by providing a callback

Table 1. Comparison between the performance (PFM) and instrumented (INST) versions of the runtime (top). Comparison between the user source code and compiler transformations for the performance and instrumented versions (bottom).

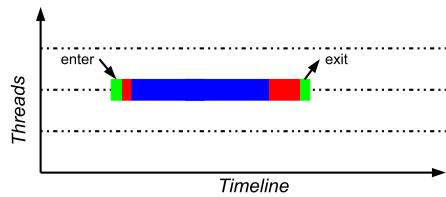
<pre> 1 task nanos_create_task(f) { 2 creating a task for f 3 } 4 nanos_submit_task(task) { 5 submitting a task 6 } </pre>	PFM runtime	<pre> 1 task nanos_create_task(f) { 2 Extrae_event(begin, "CREATING_TASK"); 3 creating a task for f 4 Extrae_event(end, "CREATING_TASK"); 5 } 6 nanos_submit_task(task) { 7 Extrae_event(begin, "SCHEDULING"); 8 submitting a task 9 Extrae_event(end, "SCHEDULING"); 10 } </pre>	INST runtime		
<pre> 1 { 2 user code 1 3 4 #pragma omp task 5 { 6 code for task A 7 } 8 user code 2 9 } 10 </pre>	User code	<pre> 1 void Outlined\$A\$(params) { 2 Extrae_event (begin, "Outlined\$A\$"); 3 code for task A 4 Extrae_event (end, "Outlined\$A\$"); 5 } 6 { 7 user code 1 8 task_t A = 9 nanos_create_task(); 10 nanos_submit_task (A); 11 user code 2 12 } </pre>	PFM user code	<pre> 1 void Outlined\$A\$(params) { 2 Extrae_event (begin, "Outlined\$A\$"); 3 code for task A 4 Extrae_event (end, "Outlined\$A\$"); 5 } 6 { 7 user code 1 8 task_t A = 9 nanos_create_task(Outlined\$A\$); 10 nanos_submit_task (A); 11 user code 2 12 } </pre>	INST user code

function. Nanos++ is then responsible for initializing the Extrae package, and also to set up the appropriate thread identifier routine by using this new API function so as the events are properly assigned to the executing thread. Once the OpenMP/OmpSs application is running, Nanos++ uses the Extrae API to emit events into the tracefile and also to notify to Extrae whether the application has finished.

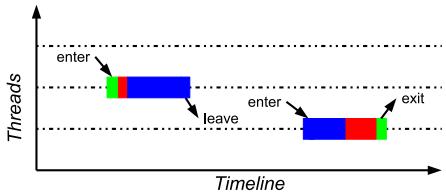
Code in the bottom part of Table 1 shows an example of use of the tasking constructs accompanied by the transformations done by the compiler when using the performance and instrumentation version of the generated code. In the instrumented version the compiler uses the Extrae API to generate two events to delimit the user function task. Also, the instrumented runtime adds events at `nanos_create_task` and `nanos_submit_task` services allowing a further analysis in the Nanos++ runtime, as shown in the top part of Table 1. For example, task creation will change thread/task state to an specific `CREATING_TASK` value, while submitting a task will change the state to `SCHEDULING`.

From the perspective of the runtime and regarding the events generated by the Nanos++, we classify them in four categories:

- *Punctual events*: a stateless event that occurs in a specific timestamp.
- *Burst events*: an event delimited by a starting and an ending timestamps (e.g. function execution).
- *State events*: it is an specific case of a burst event but due its importance it has been treated in a different manner (e.g. whether thread/task is idle).



(a) Task execution without context switches.



(b) Task execution with context switch.

Fig. 1. Execution pattern of OpenMP/OmpSs tasks

- *Point to point events*: it is a relation between two objects at different timestamps (e.g. task *a* sends data to task *b*, or task *b* is blocked because of task *a*).

Most of the generated events during instrumentation are burst or state events, which means that they are delimited by a starting point and an ending point. Task instrumentation of such kind of events requires data bookkeeping along with the cooperation of the runtime library. We refer the saved data as instrumentation context and it contains task instance instrumentation information that needs to be backed up and restored when a task is suspended or resumed, respectively. Figure 1 illustrates this problem.

Figure 1(a) shows a timeline displaying the routine that is being executed by the thread. As long as the application executes, the thread enters in three different nested routines (colored in green, red and blue, respectively), and then leaves them, returning to the previous one. If a task context switch occurs, as in Figure 1(b), it is necessary to save the state of the thread, so as it can be restored whenever it gets resumed (eventually, in a different thread).

Managing such amount of data structures during program execution would incur at additional overhead cost. So as to avoid adding overhead in the instrumentation, we have decided to move the stack management at the trace generation step. Thus the runtime library needs to register in the instrumentation library which events must be considered part of the instrumentation context. During trace generation, the process keeps track of the registered events. If the trace generation process encounters a suspend or resume event, it will backup or restore the instrumentation context in the tracefile.

The nature of the unstructured parallelism due to the usage of the tasking constructs, and more precisely in the case of the untied tasks and the existence of dependencies in OmpSs, makes the task migration and task dependency tracking

also important to be visualized. So as to fulfill this need, we have extended the API of Extrae by introducing two new event types so as to provide a mechanism that creates a relation between two threads (for example when a task is suspended in a thread and resumed in another thread). The new API calls emit information that is later translated by the tracefile construction. For the tracefile construction, we reuse the matching mechanism that Extrae has for MPI point-to-point operations in order to provide these new task relationships.

We have also implemented a mechanism that performs a deferred emission of events. This mechanism allows queuing events into a task which will be emitted when the task starts executing. Deferred events allow the runtime to place events into the tracefile without knowing when they will actually happen. One example of this usage involves task dependency tracking. During execution, a task only knows its successor tasks but does not know its predecessor tasks. Furthermore, when a task gets started it does not know whether it had predecessors or not. So the predecessor is the responsible for tracking the dependency between them without knowing which thread will execute the successor task. So as to track the dependency, the predecessor issues two events. While the first event is generated in its own instrumentation context, the second event is generated in the instrumentation context of the successor task. These two events are meant to be combined in the Paraver tracefile so as to generate a point-to-point event, and thus, correlate the two threads involved in the dependency.

3.2 Displaying OpenMP and OmpSs Tasking Traces

The Paraver resource mapping has fulfilled the needs of many types of executions, including multiple hybrid executions like MPI and OpenMP at the same time. However, adding support for tasks poses a difficulty to understand the logical execution of the threads because tasks may express irregular parallelism which is executed at any thread. Also, for the particular case of the untied tasks, a task can migrate from one thread to another, which turns that we would need an additional resource level to support representing a full-fledged execution. An example of this difficulty is shown in Figure 2(a), where the timeline shows information of the sort benchmark included in the Barcelona OpenMP Task Suite compilation [11]. In this Figure we see more than a hundred tasks (each colored differently) when the benchmark has been executed using eight threads. This Figure shows the reader the difficulty of following a set of tasks and establish their hierarchy. However, it is possible by using the visualization tool to track a particular set of tasks by selecting them by hand, as we depict in Figure 2(c). In the Figure we show in the execution of a particular thread including its migrations and its children tasks using the original display mode. From the picture we observe that the blue task was created at the first thread, it first executed at the second thread, then moves to the seventh thread and it finished its execution at the third thread. The blue task creates six tasks, four to execute the sort phase and two to execute the merge phase of the benchmark.

In order to mitigate this problem, we are experimenting with a new visualization mechanism to help on the task analysis. While the original timeline displays

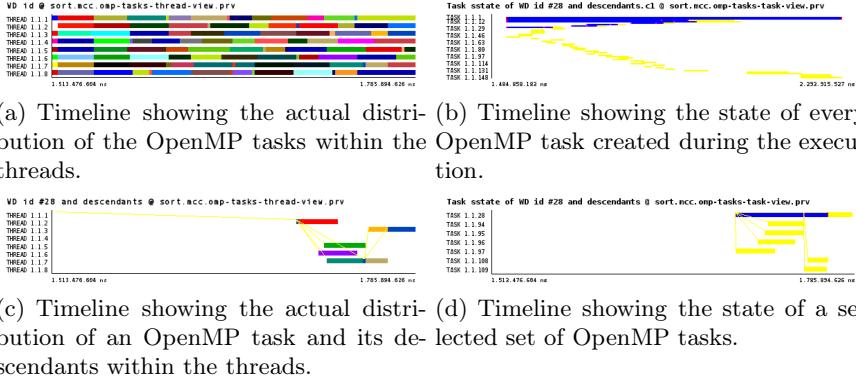


Fig. 2. Comparison of the two timeline display modes for the very same execution

information in terms of the application resources (i.e. every line in the Y-axis represents a combination of application, process and thread), the new visualization renders information in terms of tasks within a process (i.e. every line in the Y-axis represents a combination of application, process and task). For example, the whole execution of the sort benchmark by using the new display mechanism is shown in Figure 2(b). In this Figure we see the evolution of every task in the whole execution and their state. Considering that yellow means running and blue means blocked, we observe that there are two main running regions. First region is shown as a diagonal from the upper left to half-bottom right, which represents the sort phase in the benchmark. Second region is shown as a small diagonal (almost horizontal in the Figure) from the half-bottom right to bottom right, representing the merge phase in the benchmark. When both phases finalize, the main task (shown in the first row) finalizes. Figure 2(d) shows the state of the blue task referred previously in Figure 2(c) (identified as 1.1.28) and its descendants (1.1.94 to 1.1.97 for the sort phase and from 1.1.108 to 1.1.109 for the merge phase). As opposite Figure 2(c) we observe in the Figure 2(d) that task 1.1.28 spawns tasks 1.1.94 to 1.1.97 (colored in dark green, light green, purple and red, respectively, in Figure 2(c)), then waits for their completion, then spawns tasks 1.1.108 and 1.1.109 (colored in light brown and orange, respectively, in Figure 2(c)) and finally waits for their finalization.

We think that both display modes complement each other. While the original display mode is useful to determine the proper usage of the threads and task migrations, the new visualization mechanism can display more naturally the logical execution of tasks and their dependencies. A detailed application analysis would probably need using both of them so as to understand exactly which is the application behavior.

4 Experimental Results

We have done a twofold experiment of the environment we have presented. In the first experiment we evaluate the performance penalty of using the

instrumentation mechanism in Nanos++. To perform this evaluation, we use a set of benchmarks that use unstructured parallelism by using the OpenMP task clause and an application named Hydro [16]. In the second experiment we do a performance analysis of the Hydro application to show which kind of analyses can be done by using the integrated environment.

All the experiments described in this section have been executed on a system containing four hexa-core Intel® Xeon® E7450 processors running at 2.4GHz and 48 GBytes of RAM. The applications have been compiled using a development branch of the Mercurium version 1.3.5.8 and Nanos++ version 0.7a, which relies on the GNU C compiler version 4.3.4, using `-O3` as optimization flags. For the particular case of Hydro, we have compiled the application using OpenMPI version 1.4.4.

4.1 Overhead Analysis

So as to analyze the instrumentation overhead of our integrated solution, we have chosen some benchmarks from the Barcelona OpenMP Task Suite (BOTS). This compilation of benchmarks is focused in testing such unstructured parallelism in combination with other OpenMP worksharing constructs or recursive techniques. Most of them include a cut-off mechanism that is used not to generate more tasks and continue executing in a serial fashion. The cut-off allows to handle task granularity (i.e. the amount of work executed by each task) allowing users to manage a trade-off between application parallelism and the runtime overhead.

Due to space restrictions we have selected a subset of the BOTS benchmarks. This subset includes:

- *SparseLU* computes a LU matrix factorization over sparse matrices. A first level matrix is composed by pointers to smaller submatrices (blocks) that may not be allocated. In each of the SparseLU phases, a task is created for each block of the matrix that is allocated. We have used a matrix of $50 * 50$ blocks, and several block sizes from $25 * 25$ up to $200 * 200$. Due to space restrictions, we show the results for blocks of $50 * 50$ and $200 * 200$.
- *Floorplan* kernel computes the optimal floorplan distribution of a number of cells (each cell with its own shape description) which has been provided as a parameter. The algorithm calculates the minimum area size which includes all cells through a recursive branch and bound search. Floorplan application implements a cut-off based on the depth of the tree. In our experiments we have tested 20 cells input with three cut-off values (4, 5 and 6).

Table 2. TLB miss ratio of NQueens with different cut-off values

	.00-.01	.01-.02	.02-.03	.03-.04	.04-.05	.05-.06	.06-.07	.07-.08	.08-.09	.09-.10
Cut-off set to 2	99.99%	0.01%								
Cut-off set to 3	99.99%	0.01%								
Cut-off set to 4	99.99%	0.01%	0.00%							
Cut-off set to 5	44.02%	8.51%	8.55%	10.76%	11.52%	6.94%	4.30%	2.75%	1.69%	0.96%

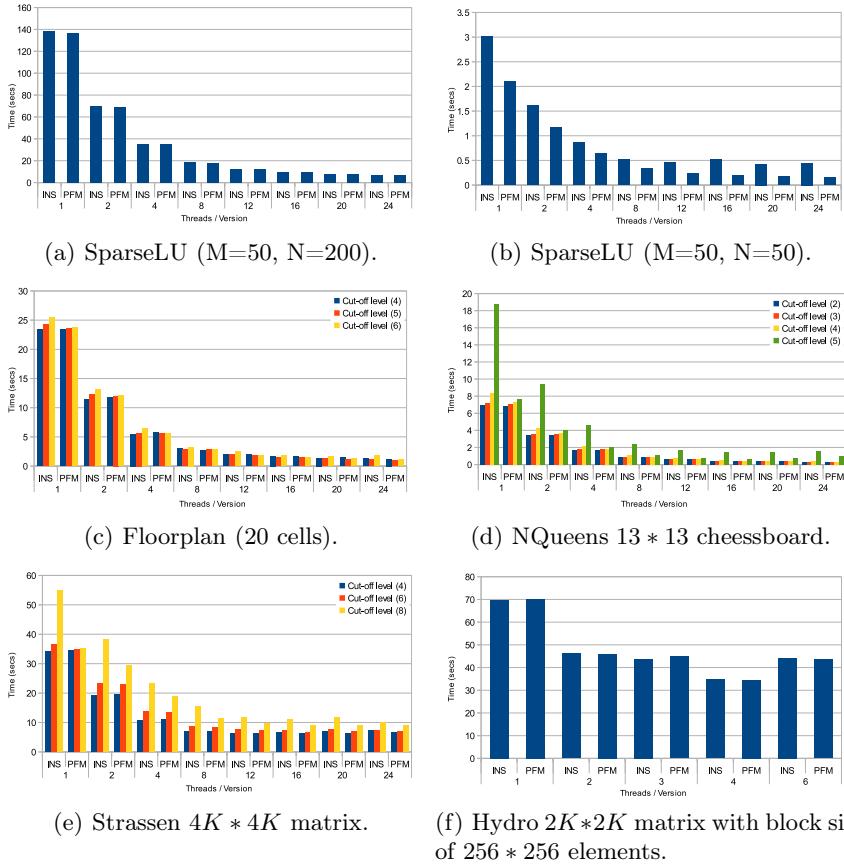


Fig. 3. Execution time for instrumented (INS) and performance (PFM) version of different applications

- *NQueens* computes all solutions of the n-queens problem, whose objective is to find a placement for n queens on an $n * n$ chessboard such that none of the queens attack any other. The benchmark uses a backtracking search algorithm with pruning, creating a task for each step of the solution. The algorithm has a cut-off mechanism based on the recursion level of the kernel. In our experiments we have tested a chessboard of $13 * 13$ and four cut-off values (2, 3, 4 and 5).
- *Strassen* algorithm uses hierarchical decomposition to multiply large dense matrices. Decomposition is done by dividing each dimension of the matrix into two sections of equal size and a task is created for each decomposition step. In our experiments we have used a matrix $4K * 4K$ of complex numbers and three cut-off levels (3, 4 and 5).

Figure 3 shows the results obtained when executing the aforementioned BOTS benchmarks plus the Hydro application. The Figure shows the comparison of

the application times by using the standard performance and the instrumented versions. The X-axis shows the number of threads and also the runtime version, while the Y-axis shows the execution time. The plots also show the execution time with different cut-offs where applicable.

In all cases, the reader can see that the instrumented version adds some overhead with respect the performance one. Overhead depends on the task granularity, which is related also with cut-off values as in Floorplan, NQueens and Strassen, or block size like in SparseLU. While on large task granularity the overhead is less than 1% we find that reducing the task granularity increases the overhead. For example in Figure 3(b) we see that the overhead for SparseLU is about 43% with 1 thread, in Figure 3(e) we observe up to a 56% and Figure 3(d) shows the largest overhead by using 1 thread (243%).

So as to understand the reasons of the large overhead experienced in the NQueens benchmark, we have executed an instrumented version of the benchmark using one thread with the different cut-off values and adding hardware performance counters metrics into the tracefile. Table 2 shows on each row the TLB miss ratio analysis (i.e. number of TLB misses per instruction) for each cut-off value shown in Figure 3(d). Each column in the table represents the TLB miss ratios ranging from 0 to 0.1 in buckets of 0.01. The value in each cell is the percentage of time that the thread has been experiencing that TLB miss ratio. Using a cut-off level of 2 we observe that 99.99% of the total time the number of TLB misses has been between 0 and 0.01, whereas the remaining time a TLB miss ratio between 0.01 and 0.02. By comparing cut-off levels between 2 and 4 we do not observe a fluctuation on this behavior. This is no longer true when the cut-off value is set to 5 where 44.02% of the total time experiences a TLB miss ratio between 0 and 0.01. More than 25% of the execution time shows a TLB miss ratio between 0.03 and 0.06. This large proportion of TLB misses per instruction is clearly increasing the overhead of the instrumentation. Our preliminary analysis of the situation indicates that the issue may appear because the large number of tasks created consumes more memory pages because of the generation of the events.

4.2 Hydro Analysis

The Hydro application is a proxy benchmark of the RAMSES [4] application, that solves a large scale structure and galaxy formation problem using a rectangular 2D space domain split in blocks. Hydro is a mixed application that combines OmpSs tasking constructs with data dependencies and MPI. The application input requires to execute using 4 MPI processes and works with matrices of $2K * 2K$ elements split in blocks of $256 * 256$ elements.

First, we have analyzed the overhead of the instrumentation which is plotted in Figure 3(f). The diagram shows a minimal overhead, typically close to 1% and that the application does not scale linearly in respect to the number of threads used. In fact, the application does not even scale when using more than four threads.

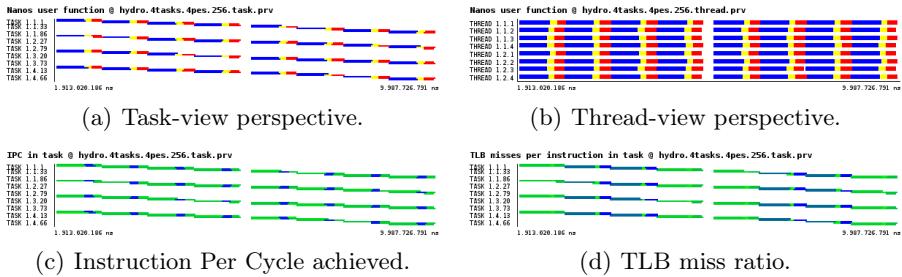


Fig. 4. Different time-line perspectives of the Hydro execution

Due to space restrictions, we study the case that gives better performance from those shown in Figure 3(f). In Figure 4(a) we show a portion of the whole application execution in a timeline using the task view. Within the Figure three different types of OmpSs tasks appear colored in blue, yellow and red, which correspond to three outlined routines generated by the Mercurium compiler that are executed as OmpSs tasks. We observe that the yellow tasks do not start until the blue tasks finish and that the red tasks do not execute until the yellow tasks finish, this is because of the data dependencies in the source code. Although data dependencies limit the concurrency of the application because blue, red and yellow tasks do not overlap in time, we observe in Figure 4(b) that 4 instances of tasks are executed in parallel. This parallelism exists because the data dependencies in the loop are intra-iteration but not inter-iteration, giving the runtime the possibility to execute up to four iterations at a time.

Figures 4(c) and 4(d) show the achieved Instructions Per Cycle (IPC) and the TLB miss ratio, respectively. The data in these timelines are colored by a gradient that goes from green to blue, where green represents the lowest value and blue represents the highest value. We can correlate Figure 4(a) with Figure 4(c) and we can note that the tasks coded in yellow are the task achieving the highest IPC (0.57) whereas the rest of the code task runs at much slower IPC (0.13). By comparing Figures 4(a) and 4(d) we notice also that the yellow task has the lowest TLB miss ratio (less than 1 TLB miss per kilo-instruction). It is also noteworthy that the red and blue tasks present two different behaviors with respect to TLB misses. Half invocations of these tasks experience a high number of TLB misses (about 80 TLB misses per kilo-instruction in the red task, for instance) whereas the other do not miss so much in the TLB (less than 1 TLB miss per kilo-instruction again in the red task). These tasks execute the functions named `updateConservativeVars` and `gatherConservativeVars`, which depending on one of its parameters traverses a matrix by its leading dimension or not. Those invocations of the routine that do not traverse the matrix by its leading dimension present the highest miss rates in the TLB.

5 Related Work

POMP [20] is an extension to OpenMP proposed by Mohr *et al.*. POMP extensions include different mechanisms to obtain information from the OpenMP runtime, mostly using callback functions. POMP issues these callbacks whenever the OpenMP runtime reaches a selected specific code locations like: fork and join threads, invoking a parallel outlined function, entering and leaving a barrier, among others. To our knowledge, POMP has been fully implemented by Sun Microsystems[15] and partially implemented by IBM [10]. POMP is currently incomplete [10], *i.e.* not everything needed for a working implementation is specified. Compared to the work we present, it differs from POMP in the sense that the parallel programming runtime is responsible for emitting events through the instrumentation library.

Scalasca [24], Vampir [21] and TAU [23] are performance tools that collect data from OpenMP applications, among other types of applications. OpenMP data collection in these tools can be done through OPARI [19], DynInst [8]. OPARI implements POMP by performing a source to source translation of the user code, and thus limits the optimization opportunities of the application and also presents some other drawbacks as making implicit barriers explicit and not being able to instrument the worksharing constructs. DynInst is a binary rewriter library that allows adding monitors in specific locations of the code. The developer of the tool needs to know the OpenMP runtime calls, and this is not always possible. Also, each OpenMP manufacturer has its own runtime, which limits the chances of instrumenting all the OpenMP applications. The solution we propose relies on a tight cooperation between the parallel programming model and the performance analysis tool, this way the tool does not only know information about when events (like entering or leaving parallel constructs) occur but also information of the internal runtime state and also migration information of untied tasks. Also, the usage of Paraver allows a more flexible and precise analysis of the performance data.

Fürlinger *et al.* have written about a performance profiling for OpenMP 3.0 in [13]. They use a more recent version of OPARI, OPARI2 [18], to instrument OpenMP applications that use OpenMP tasking constructs. In their work, they provide a summary of the time spent on each task construct, the function executed as a such, in addition to imbalance, overhead and synchronization time. Because of the nature of the profile, it provides a summary of the execution and it cannot provide the rich details of the execution we can show by using a trace-file and a visualization tool as Paraver. Moreover, although OPARI2 provides support for OpenMP tasks, to our knowledge other performance tools have not extended their visualizations tools so as to display the execution of the OpenMP tasks.

Finally, ROSE [17] is an open source compiler infrastructure to build source-to-source program transformations and analysis tools. Authors proved their system by providing a transformation interface for the OpenMP programming model that tested lock misuse by replacing runtime library calls. Although the work is only developed for runtimes of the GNU and Omni compilers, it could

probably be extended to support other runtimes and to generate event trace based performance data. However, we believe that the emitting performance information of the runtime within it, can provide richer analysis rather than information gathered at runtime library calls.

6 Conclusions and Future Work

In this paper we have summarized the integration of a parallel runtime library that supports tasking constructs and an instrumentation library. Mercurium and Nanos++ work together to support OpenMP Tasking model and they also can produce all the information needed by instrumentation which will be used by Extrae to generate a Paraver trace. The close cooperation between them allows to generate highly detailed information about the program execution including information from the programming model perspective and also from the runtime internal information.

We have also presented a new trace display based on task, which complements the classic view by showing the application execution from the task perspective. This new view mode allows us to follow the liveness of a certain task through timeline without taking into account how it switches from one thread to another. Using task view mode we can easily follow task dependencies and we can also combine it with other existing events (e.g. hardware counters) to correlate them, as we have shown in the Hydro application study.

We have analyzed the overhead and we have studied the impact of our instrumentation in several application kernels and using different configurations. We are aware about the current limitations on terms of overhead of our implementation. As of today, we are gathering a huge amount of information from the runtime including: the number of ready tasks, the number of tasks blocked, the number of yields executed and the idle loop executions among others. We think that gathering all such information will easily exceed the capabilities of the analysis tool for large runs and we would like to tackle this issue by rethinking which of the generated events are really required and which could be optional. Additionally, we are also working on a version of Nanos++ for distributed-memory systems [9] and accelerators [12]. We plan to extend the instrumentation mechanism in order to support these versions of the runtime.

Acknowledgments. We thankfully acknowledge the support of the European Commission through the Mont-Blanc project (FP7-288777), and the coordinated twin project HOPSA (FP7-ICT-2011-EU-Russia grant number FP7-277463, and Russian Ministry of Education and Science number 07.514.12.4001), and the support of the Spanish Ministry of Education (TIN2007-60625, CSD2007-00050) and the Generalitat de Catalunya (2009-SGR-980).

Intel, Xeon, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

* Other brands and names are the property of their respective owners.

References

1. Extrae instrumentation package, <http://www.bsc.es/paraver> (accessed April 2012)
2. Mercurium C/C++ source-to-source compiler, <http://pm.bsc.es/projects/mcxx> (accessed May 2012)
3. Nanos++RTL. <http://pm.bsc.es/projects/nanox> (accessed May 2012)
4. RAMSES, <http://web.me.com/romain.teyssier/Site/RAMSES.html> (accessed May 2012)
5. Top 500 supercomputing sites, <http://www.top500.org> (accessed June 2012)
6. Ayguade, E., Badia, R.M., Cabrera, D., Duran, A., Gonzalez, M., Igual, F., Jimenez, D., Labarta, J., Martorell, X., Mayo, R., Perez, J.M., Quintana-Ortí, E.S.: A Proposal to Extend the OpenMP Tasking Model for Heterogeneous Architectures. In: Müller, M.S., de Supinski, B.R., Chapman, B.M. (eds.) IWOMP 2009. LNCS, vol. 5568, pp. 154–167. Springer, Heidelberg (2009)
7. OpenMP Architecture Review Board. OpenMP Application Program Interface v 3.0 (May 2008)
8. Buck, B., Hollingsworth, J.K.: An API for runtime code patching. Int. J. High Perform. Comput. Appl. 14(4), 317–329 (2000), <http://www.dyninst.org>
9. Bueno, J., Martinell, L., Duran, A., Farreras, M., Martorell, X., Badia, R.M., Ayguade, E., Labarta, J.: Productive Cluster Programming with OmpSs. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011, Part I. LNCS, vol. 6852, pp. 555–566. Springer, Heidelberg (2011)
10. De Rose, L., et al.: An Implementation of the POMP Performance Monitoring Interface for OpenMP Based on Dynamic Probes, <http://www.research.ibm.com/actc/projects/pdf/T16p.pdf> (accessed May 2012)
11. Duran, A., et al.: Barcelona OpenMP Tasks Suite: A set of benchmarks targeting the exploitation of task parallelism in OpenMP. In: International Conference on Parallel Processing, ICPP 2009, pp. 124–131. IEEE (2009), <https://pm.bsc.es/projects/bots> (accessed May 2012)
12. Ferrer, R., Planas, J., Bellens, P., Duran, A., Gonzalez, M., Martorell, X., Badia, R.M., Ayguade, E., Labarta, J.: Optimizing the Exploitation of Multicore Processors and GPUs with OpenMP and OpenCL. In: Cooper, K., Mellor-Crummey, J., Sarkar, V. (eds.) LCPC 2010. LNCS, vol. 6548, pp. 215–229. Springer, Heidelberg (2011)
13. Fürlinger, K., Skinner, D.: Performance Profiling for OpenMP Tasks. In: Müller, M.S., de Supinski, B.R., Chapman, B.M. (eds.) IWOMP 2009. LNCS, vol. 5568, pp. 132–139. Springer, Heidelberg (2009)
14. Hempel, R.: The MPI Standard for Message Passing. In: Gentzsch, W., Harms, U. (eds.) HPCN-Europe 1994. LNCS, vol. 797, pp. 247–252. Springer, Heidelberg (1994)
15. Itzkowitz, M., et al.: An OpenMP Runtime API for Profiling, <http://www.comppunity.org/futures/omp-api.html> (accessed May 2012)
16. Lavallea, P.F., et al.: HYDRO, <http://www.prace-ri.eu> (accessed May 2012)
17. Liao, C., Quinlan, D.J., Panas, T., de Supinski, B.R.: A ROSE-Based OpenMP 3.0 Research Compiler Supporting Multiple Runtime Libraries. In: Sato, M., Hanawa, T., Müller, M.S., Chapman, B.M., de Supinski, B.R. (eds.) IWOMP 2010. LNCS, vol. 6132, pp. 15–28. Springer, Heidelberg (2010)

18. Lorenz, D., Mohr, B., Rössel, C., Schmidl, D., Wolf, F.: How to Reconcile Event-Based Performance Analysis with Tasking in OpenMP. In: Sato, M., Hanawa, T., Müller, M.S., Chapman, B.M., de Supinski, B.R. (eds.) IWOMP 2010. LNCS, vol. 6132, pp. 109–121. Springer, Heidelberg (2010)
19. Mohr, B., et al.: Design and Prototype of a Performance Tool Interface for OpenMP. *The Journal of Supercomputing* 23, 105–128 (2001), doi:10.1023/A:1015741304337
20. Mohr, B., et al.: A Performance Monitoring Interface for OpenMP. In: Proceedings of the Fourth Workshop on OpenMP, EWOMP 2002 (2002)
21. Nagel, W.E., et al.: VAMPIR: Visualization and analysis of MPI resources. *Supercomputer* 12(1), 69–80 (1996)
22. Pillet, V., et al.: Paraver: A tool to visualize and analyze parallel code. *Transputer and Occam Developments*, 17–32 (April 1995), <http://www.bsc.es/paraver> (accessed April 2012)
23. Shende, S.S., Malony, A.D.: The TAU parallel performance system. *Int. J. High Perform. Comput. Appl.* 20(2), 287–311 (2006)
24. Wolf, F., et al.: Usage of the SCALASCA for scalable performance analysis of large-scale parallel applications. In: Tools for High Performance Computing, pp. 157–167. Springer, Heidelberg (2008)

Strategies for Real-Time Event Reduction

Michael Wagner and Wolfgang E. Nagel

Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden, 01062 Dresden, Germany
michael.wagner@zih.tu-dresden.de

Abstract. One of the most urgent issues in event tracing is the number of resulting event trace files pushing against the limits of today’s parallel file systems. To address this issue, we present strategies for real-time event reduction, which guarantee that data of an event tracing measurement fits into a single memory buffer. Therefore, they are a key step towards a complete in-memory event tracing workflow enabling event trace analysis on very high scales without the limitations of today’s parallel file systems. In addition, we define criteria to compare different reduction strategies and evaluate their benefits. Furthermore, we show how traditional memory buffering can be enhanced to realize these strategies with minimal overhead.

Keywords: Event Reduction, In-Memory Event Tracing, Trace Format.

1 Introduction

Development of efficient applications is a challenging task. It demands knowledge of the underlying hardware, compilers and the own source code, which is not trivially given, especially for huge and complex applications. The massive parallelism of today’s HPC (High Performance Computing) systems creates another dimension of complexity. Developers have to consider parallel programming paradigms, communication, and load balancing issues to utilize the enormous parallel computing resources. Therefore, the development of efficient parallel software benefits greatly from appropriate supporting tools such as performance analysis tools.

Performance analysis tools collect information about the application behavior and help the developer to better understand his code and to identify performance problems. The two main approaches are profiling and event tracing. While profiling is the gathering of summarized information about different performance metrics, event tracing records runtime events together with a precise time stamp and further event specific metrics. Thus, event tracing collects very detailed information about an application, which reveals a deep insight into its behavior, e.g. communication patterns or load balancing behavior. On the downside, event tracing usually results in huge data volumes since the number of recorded events is quite large. In fact, this is one of the most urgent challenges: the enormous amount of data that is recorded in a single measurement run. Since the recorded

data is typically stored in one file per process or thread, especially, the increasing number of resulting files is already pushing against the limits of today's parallel file systems. Without any special treatment, it is possible to handle about ten or twenty thousand of parallel processes. There are already some solutions to circumvent current file system limitations and handle hundreds and thousands of processing elements [2,4]. But, what about future systems with millions or tens of millions of processing elements?

Keeping the data in main memory for the complete workflow from measurement to analysis would completely circumvent file system interactions and, therefore, the file system's limitations. Moreover, it would eliminate the overhead of file creation and writing altogether. In addition, this would enable new ways of event tracing. Our target is to enable *interactive event tracing*. This new workflow would provide faster feedback from the measurement: a measurement could start with a specified set of tracing parameters and interrupt at a defined point. Then, the developer gets a first insight into the application behavior and can decide how to go on: delete unnecessary parts of the measurement, modify the level of detail, or maybe start over because the application is running on a wrong data set. This is just a small glimpse; much more can be done with this approach (see [9]).

But there is one catch: to avoid file interaction, the recorded data has to fit into a single memory buffer. Unfortunately, this is not given for most applications. Quite contrary, a measurement can collect hundreds of megabytes up to gigabytes of data per process. To make things worse, most applications utilize main memory quite well. Thus, the part of the main memory used to store event data should be rather small.

Hence, the challenge is to fit all the data into a single memory buffer. We think of three major steps to achieve this. The first step is a combination of automated intelligent high-level filters, phase based selection and user defined filtering. Currently, we study the effects of runtime loop phase selection, i.e. storing only distinctive representatives of a loop class. There are many different filters and selections to study and we are convinced, that combined they can achieve a very good reduction of "unnecessary" events. The second step is an efficient storage of events in the memory buffer. In [9] we describe our efforts in this area. We were able to increase memory efficiency during runtime by a factor up to 5.8 without increasing the overhead of the event tracing library. In this paper, we focus on the third and last step; a very important one. The two first parts can reduce the memory allocation remarkably but only by a limited factor, i.e. they cannot guarantee that the data fits into a single memory buffer. The third and last part has to ensure that all data fits – always. Therefore, this step is completely different to the filtering and compression steps. It is triggered in the moment the memory buffer is exhausted; typically this is the point where the memory buffer is either flushed to a file or the measurement is aborted. The challenge is to transparently make space available again by reducing the events stored in the memory buffer. This event reduction must be done with minimal overhead. We call this step *real-time event reduction*.

For a better understanding of the complexity of this last step we must distinguish between high-level information and low-level information. The measurement environment holds a lot of information about the application in its internal data structures. Thus, high-level filters are best located in the measurement environment. The memory buffer management is typically delegated to an event tracing library. This provides many benefits for the event tracing tools like an optimized data compression and interoperability with different analysis tools. But, this also means that at the point where the event reduction step is located (the event trace library), only a subset of low-level information like event classes is available. So, the crux of the event reduction step is to achieve reduction with minimal overhead and minimal information demand.

The contribution of this paper is the presentation and comparison of different new approaches to achieve real-time event reduction and, therefore, guarantee to always fit all recorded data into a single memory buffer. Real-time event reduction is a key step towards a complete in-memory event tracing workflow, which enables event trace analysis on very high scales without today's file system limitations. In addition, we show how these different approaches can be realized. As far as we know, this is novel work and there are no realization strategies presented so far.

The approaches and realization strategies are based on the Open Trace Format 2 (OTF2) [1], a state-of-the-art Open Source event trace library used by the performance analysis tools VAMPIR [6], SCALASCA [3], and TAU [8]. The methods can also be generalized on other event based tracing libraries.

In the following section we distinguish our work from other current approaches. In section 3 we describe the different approaches for real-time event reduction in detail and show realization strategies in section 4. At the end, we summarize the presented work and give an outlook on our future work.

2 Related Work

Since huge numbers of files are an urgent problem in current parallel file systems, a number of potential solutions are in recent development. Two approaches used in event tracing are SIONlib [2] and IOFSL [4]. SIONlib merges multiple logical files into a single or a few physical files. A merged SIONlib file consists of huge pre-allocated segments for the logical file handles; using the file system's capability to handle large sparse files. The I/O Forwarding Scalability Layer (IOFSL) provides an atomic append capability that can be used by applications that have huge parallel per process output like tracing. This allows to write output from many processes into a single or fewer files without any coordination between the processes. Each process also knows the offset where its own data is stored, enabling data access without parsing the entire shared output file.

An alternative is to avoid file interaction at all. While the general idea of keeping all data in main memory for the whole workflow is not a new one, to the best of our knowledge, the strategies of how to do so are not presented so far; except the very basic idea of Section 3.1, which is used i.a. in [3,6].

3 Reduction Strategies

In this section we present three different strategies for real-time event reduction. While these strategies may seem rather simple, we must keep in mind that at the point these strategies operate there is only a limited view on the application's total configuration (see Section 1). In an event tracing library, where the memory buffer is located, there is no information about the call-tree, other processes, or the context of a single event. The strategies need to work with minimal information like the type of an event. Therefore, it is obvious that these strategies cannot be compared with high-level filtering techniques. The main goal of the reduction strategies is to guarantee that there is never an overflow of data in the memory buffer. Or in other words, only a single memory buffer is necessary to record an entire measurement.

The basic idea is to get a coarse overview of the application behavior in any case. Especially at a first measurement the user has only limited knowledge and understanding of the application's behavior. So, user based high-level filtering might be less effective, and, therefore, the main reduction is done with event reduction strategies. The resulting overview of the application's behavior is coarse. But the user gets an overview at all and is able to derive first knowledge about the application behavior. The gained knowledge can then be used to achieve better user based high-level filtering and, with this, a more accurate overview of the application behavior.

Another aspect that needs to be considered is the timing of the reduction. While high-level filtering reduces during the whole runtime, event reduction engages at the moment the buffer is exhausted, i.e. a request to store the next event is issued but there is no space left in the memory buffer. At this moment the reduction strategy intervenes, reduces the amount of already stored events, and makes memory space available again. This reduction must work with very little overhead, so that storing the event is delayed as little as possible and, therefore, the recorded application behavior is minimally disturbed.

These two requirements, minimal overhead and minimal information demand, must be met by all reduction strategies. Thus, these criteria are not feasible to compare the different reduction strategies. For comparison we use the resulting data that is left after reduction. But this can not be evaluated on a quantitative basis. Rather, we must find criteria to review the quality of the remaining information. Performance analysis has two main goals: first, to better understand the application behavior and, second, to identify performance problems. Therefore, our evaluation is based on how good these goals can still be achieved with the reduced data set: is it still possible to understand the application behavior and is it still possible to detect occurring performance problems? Next to that, we analyze the granularity of reduction steps. This means, how much data is discarded in a single reduction step. If the steps are too huge, a lot of data is lost in a single reduction step, even if not necessary.

In the following we present and evaluate three different reduction strategies: reduction by order of occurrence, by event class, and by call stack level.

3.1 Reduction by Order of Occurrence

The simplest strategy is the reduction by order of occurrence, i.e. to stop recording once the memory buffer is exhausted. This provides a complete recorded application behavior from the start until the point recording stopped. After that point there is no information about the application behavior. Thus, a good understanding can be obtained about the recorded part of the application but there is no information about the application behavior after the recording stopped. For the detection of performance problems applies the same condition. A performance problem that occurs in an early part of the application is analyzable without any constraints because the complete event stream of this part is available. Performance problems occurring in a late state of application runtime are not detectable at all since there are no events recorded after the memory buffer is exhausted. Therefore, the quality of the overall information about the application strongly depends on the structure of the application. Since it is not possible to identify beforehand, where performance problems will occur or where the parts are that need better understanding (because this is the outcome of performance analysis), in most cases this strategy delivers poor results. To analyze a specific code region, a program phase selection on a higher level (see Section 1) delivers best results. However, this already implies a good understanding of the application. Nevertheless, the event distribution by occurrence has a very high granularity; an event-wise reduction is possible.

3.2 Reduction by Event Class

The single events that are recorded by event tracing can be categorized in different classes, e.g. entering and leaving a code region, peer-to-peer and global communication, performance metrics like hardware counters, and file interaction. Naturally, not all classes of events are of the same importance to analyze an application's behavior. For example, if a developer wants to analyze the communication behavior, communication events are very important while hardware counters are less important. When looking into single thread performance it is the other way around. Thus, it is possible to order the classes of event types by importance and start reduction with the least important event classes. Since there is no generally meaningful order for all applications, this order should be specified by the user. This strategy can provide a good knowledge about the application behavior represented by the remaining event types. About application behavior deducible by the reduced event types no knowledge can be obtained at all. This also applies for the detection of performance problems. Performance problems that can be recognized by the remaining event classes can be fully analyzed because all events are available. But, performance problems deducible by the discarded event classes can not be detected. In addition, performance problems that only can be derived by a combination of two or more event classes cannot be detected when one of the event classes has been reduced. Hence, the quality of the overall information about the application depends mainly on an appropriate order of event classes by their importance.

Unfortunately our statistical analysis shows that there are only three dominating event classes: enter/leave events, performance counters, and MPI peer-to-peer events (see Figure 1). All other event classes have only a marginal fraction of the total memory allocation. Therefore, this strategy has a limited potential for reduction since the event distribution by event type is coarse. Nevertheless, this strategy can be a good choice to sort out events of one of the main classes if they are of less importance.

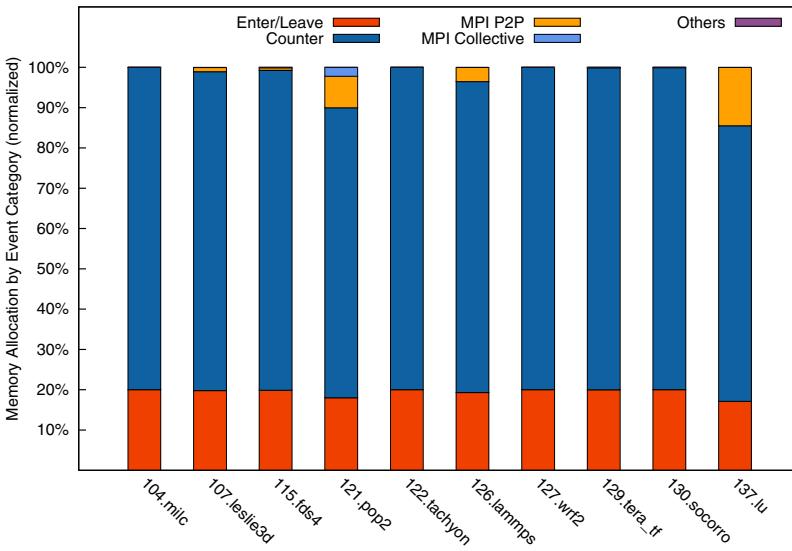


Fig. 1. Memory allocation by event type for SPEC MPI 2007 applications. There are only three dominating event classes: enter/leave events, hardware performance counters, and MPI peer-to-peer events. All other event classes have only a marginal fraction of the total memory allocation.

3.3 Reduction by Call Stack Level

A third strategy is to select events for reduction by their call stack level, i.e. events with the highest call stack level are first to be reduced¹. This strategy delivers different results according to understanding of application behavior and detection of performance problems than the first two strategies. Of course, detailed information about events of discarded call stack levels is lost, too. But information is still partly available in lower call stack levels. While in the other

¹ Since MPI does not provide any features to determine matching MPI calls (e.g. a receive operation to its according send operation) automatic message matching is an integral part in event trace analysis. But, correct automatic message matching can only be done, if all participating MPI calls are recorded. Therefore, MPI calls are treated separately in this strategy, i.e. either all MPI calls of all levels are kept or all are discarded.

two approaches, for reduced application sections it is not possible to identify a performance problem at all, this strategy provides at least a hint for a potential performance problem. When reducing the call stack level containing events carrying information about a performance problem, the cause of the performance problem is lost but it is still possible to recognize the impact of the performance problem. For example in Figure 2 there is a load imbalance caused by the function *bar* on process two. With a reduction of the call stack level containing *bar*, the cause is not visible anymore. But, the impact in *foo* and, therefore, the performance problem itself is still detectable and can be analyzed in detail with another measurement run focused on this specific code region. Therefore, the knowledge about the application and the ability to detect performance problems is reduced but not completely lost for individual parts.

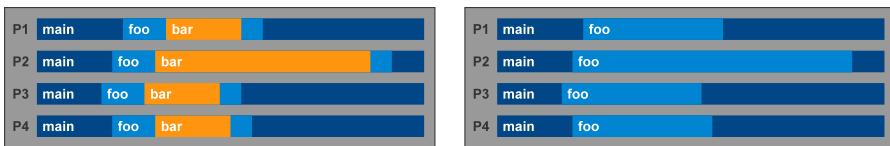


Fig. 2. Timeline representation of a sample load imbalance for four processes. Function *main* is calling *foo*, which is calling *bar*. The load imbalance is caused by function *bar* on process two. Left: the complete event stream. Right: event stream without the highest call stack level.

The applicability of this strategy strongly depends on the application design with regard to call stack level distribution. The best case is a deep and equally distributed call stack (Figure 3 top, left). This enables a reduction in very fine steps. Our statistical analysis showed that there is a variety of different call stack distributions (see Figure 3). There are some applications where a reduction can be done in several steps (Figure 3 top, right and bottom, left) and, therefore, are well suited to this reduction strategy. But, there are also applications where a reduction can only be done in a few big steps (Figure 3 bottom, right). This means, a lot of information is lost within a single reduction step and it is unlikely to utilize the memory buffer well. On the other hand, the statistical analysis showed that the reduction granularity for the analyzed applications mostly is related to the number of events, i.e. applications with a low call stack granularity, typically, produce a moderate number of events. For example, the application 115.fds4 generates an order of magnitude less events than the two other applications in Figure 3.

Altogether, a combination of the different reduction strategies provides best results. For example, when looking into load imbalances it is best to discard the communication events first. After that, it is better to reduce the call stack levels first instead of discarding the performance metric events. Such a combined strategy can utilize the benefits of the single strategies and deliver the most descriptive resulting overview of the application behavior.

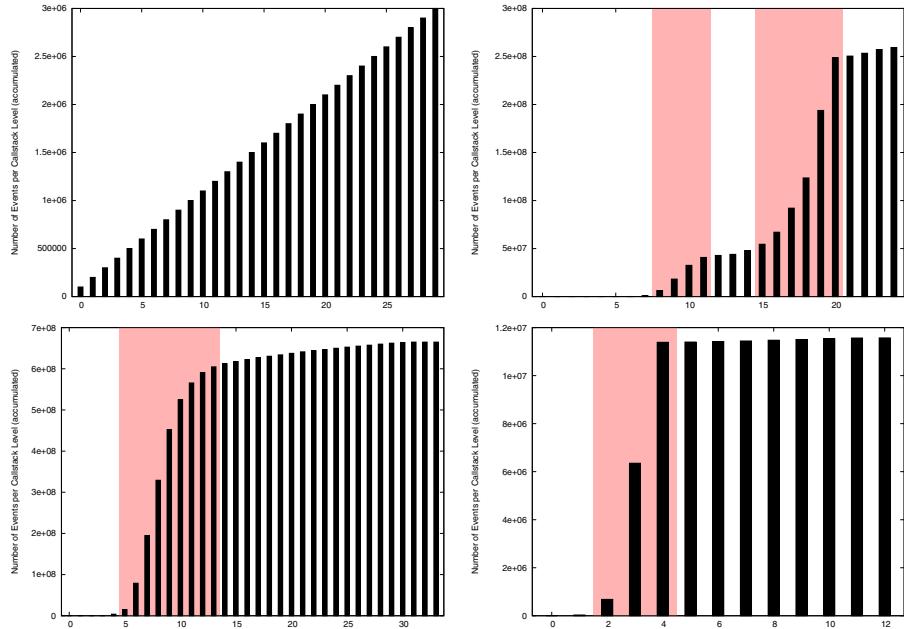


Fig. 3. Event distribution by call stack level (accumulated) for selected SPEC MPI 2007 applications. Top, left: the best case is a deep and equally distributed call stack. Top, right: 130.soccero - a reduction in 4+6 steps is possible (red zones). Bottom, left: 122.tachyon - a reduction in nine steps is possible. Bottom, right: 115.fds4 - a reduction can only be done in two big steps.

4 Realization Strategies

Typically, event trace formats store the events in the order of their occurrence within a single memory buffer per process [1,5,10]. This is the most trivial form for writing and also the natural order to read the events in again. Therefore, the first reduction strategy that reduces events by the order of their occurrence is easy to apply, e.g. just stop recording once the memory buffer is exhausted. This approach does not require significant modifications to the general event handling and generates basically no overhead.

The two other approaches are much more complex to realize. Using the typical single buffer approach means that the different event types and the events of different call stack levels are scattered over the whole memory buffer. Thus, once the memory buffer is exhausted, all events must be read to identify those who need to be discarded, e.g. those of a specific event class or the highest call stack level. This generates huge overhead and breaks the requirement for a real-time solution. Even more, after the reduction the free memory space is highly fragmented in a lot of very small slots. Hence, this method is not useful at all.

The only way to avoid this situation is to order the events from the beginning by the different reduction criteria. This means, events must be sorted by

occurrence, by event classes, and by call stack level altogether. To do so in a single memory buffer requires a partitioning of the memory buffer in many small fragments reserved for each reduction criteria e.g. one small fragment for each call stack level that might occur or each event class. In the following we use the term *reduction item* to entitle either a single call stack level or a single event class. Each fragment for the different reduction items should be utilized until the total memory usage hits the limit. In addition, after reduction the freed memory space should be available for the remaining reduction items. Obviously, this cannot be done with the classic single memory buffer approach. The distribution of memory to the different call stack levels and event classes must be organized in a highly dynamic way.

To realize this dynamic distribution, instead of one huge buffer, many very small bins (e.g. 64 kB each) are used. These bins are provided on request to the different call stack levels or event classes. Such a request is triggered either when there is no bin available for a reduction item, i.e. the first event for this reduction item should be written or the current bin is full. When there are no bins left and a request is triggered, one reduction item is reduced (e.g. the highest call stack level) and all the bins used by this reduction item are made available for the remaining reduction items; in particular, for the one that triggered the request. To avoid frequent costly allocation and freeing of memory resources a memory management system is applied. Such a memory management system as used in OTF2 by Score-P [7] allocates the complete memory resources at the beginning and delivers small memory pages from the complete memory allocation. With this, the reduction strategies introduce minimal overhead and enable a real-time event reduction.

5 Conclusion

In this paper we presented strategies for real-time event reduction. These strategies guarantee that data of an event tracing measurement fits into a single memory buffer. Therefore, they are a basic step towards a complete in-memory event tracing workflow, which enables event trace analysis on very high scales without the limitations of today's parallel file systems. In addition, we defined criteria to compare these strategies and evaluated their benefits. And, we showed how traditional memory buffering needs to be enhanced to realize these strategies with minimal overhead.

6 Future Work

The presented strategies as well as other optimizations like enhanced compression techniques [9] are based on and developed in cooperation with OTF2. Therefore, the goal is to integrate the strategies for real-time event reduction into the Open Trace Format 2 and make them available to a wide user group. Furthermore, our final goal is the realization of interactive event tracing as described before to enable faster insight into the application behavior. The implementation

of these strategies enables in-memory event trace analysis and, therefore, is a key step towards interactive event trace analysis.

References

1. Eschweiler, D., Wagner, M., Geimer, M., Knüpfer, A., Nagel, W.E., Wolf, F.: Open Trace Format 2 – The Next Generation of Scalable Trace Formats and Support Libraries. In: Proc. of the International Conference on Parallel Computing, ParCo, Ghent, Belgium (2011) (accepted for publication)
2. Frings, W., Wolf, F., Petkov, V.: Scalable massively parallel i/o to task-local files. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC 2009, pp. 17:1–17:11. ACM, New York (2009), <http://doi.acm.org/10.1145/1654059.1654077>
3. Geimer, M., Wolf, F., Wylie, B.J., Ábrahám, E., Becker, D., Mohr, B.: The Scalasca Performance Toolset Architecture. *Concurrency and Computation: Practice and Experience* 22(6), 702–719 (2010)
4. Ilsche, T., Schuchart, J., Cope, J., Kimpe, D., Jones, T., Knüpfer, A., Iskra, K., Ross, R., Nagel, W.E., Poole, S.: Enabling Event Tracing at Leadership-Class Scale through I/O Forwarding Middleware. In: Proceedings of the 21th International Symposium on High Performance Distributed Computing, HPDC 2012. ACM (June 2012)
5. Knüpfer, A., Brendel, R., Brunst, H., Mix, H., Nagel, W.E.: Introducing the Open Trace Format (OTF). In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3992, pp. 526–533. Springer, Heidelberg (2006)
6. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S., Nagel, W.E.: The Vampir Performance Analysis Tool Set. In: Tools for High Performance Computing, pp. 139–155. Springer (July 2008)
7. Knüpfer, A., Rössel, C., an Mey, D., Biersdorf, S., Diethelm, K., Eschweiler, D., Gerndt, M., Lorenz, D., Malony, A.D., Nagel, W.E., Oleynik, Y., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., Wolf, F.: Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Tools for High Performance Computing 2011, Dresden, Germany (2011) (accepted for publication)
8. Shende, S., Malony, A.D.: The TAU Parallel Performance System, SAGE Publications. *International Journal of High Performance Computing Applications* 20(2), 287–331 (2006)
9. Wagner, M., Knüpfer, A., Nagel, W.E.: Enhanced Encoding Techniques for the Open Trace Format 2. *Procedia Computer Science* 9, 1979–1987 (2012)
10. Wolf, F., Mohr, B.: EPILOG Binary Trace-Data Format. Tech. Rep. FZJ-ZAM-IB-2004-06, Forschungszentrum Jülich (May 2004)

A Scalable InfiniBand Network Topology-Aware Performance Analysis Tool for MPI*

Hari Subramoni, Jerome Vienne, and Dhabaleswar K. (DK) Panda

Department of Computer Science and Engineering,

The Ohio State University

{subramon, viennej, panda}@cse.ohio-state.edu

Abstract. Over the last decade, InfiniBand (IB) has become an increasingly popular interconnect for deploying modern supercomputing systems. As supercomputing systems grow in size and scale, the impact of IB network topology on the performance of high performance computing (HPC) applications also increase. Depending on the kind of network (FAT Tree, Tori, or Mesh), the number of network hops involved in data transfer varies. No tool currently exists that allows users of such large-scale clusters to analyze and visualize the communication pattern of HPC applications in a network topology-aware manner. In this paper, we take up this challenge and design a scalable, low-overhead InfiniBand Network Topology-Aware Performance Analysis Tool for MPI - INTAP-MPI. INTAP-MPI allows users to analyze and visualize the communication pattern of HPC applications on any IB network (FAT Tree, Tori, or Mesh). We integrate INTAP-MPI into the MVAPICH2 MPI library, allowing users of HPC clusters to seamlessly use it for analyzing their applications. Our experimental analysis shows that the INTAP-MPI is able to profile and visualize the communication pattern of applications with very low memory and performance overhead at scale.

1 Introduction

Across scientific domains, application scientists are constantly looking to push the envelope by running large-scale, parallel jobs on supercomputing systems. Message Passing Interface (MPI) [12] is a very popular programming model being used to write such large-scale parallel programs. Supercomputing systems are currently comprised of thousands of compute nodes based on modern multi-core architectures. Interconnection networks have also rapidly evolved to offer low latencies and high bandwidths to meet the communication requirements of parallel applications. InfiniBand has emerged as a popular high performance network interconnect and is being used increasingly to deploy some of the top supercomputing installations around the world. Most supercomputing systems consist of racks of compute nodes and use complex network architectures comprised of many levels of leaf and spine switches. Different factors can affect the performance of applications utilizing such IB clusters. One of these factors is the number of hops the packet has to traverse in the IB network. In [9], Hoefler et al. showed

* This research is supported in part by U.S. Department of Energy grant #DE-FC02-06ER25755; National Science Foundation grants #CCF-0916302, #OCI-0926691 and #CCF-0937842.

network topology can have a significant impact on the performance of scientific parallel applications. In our previous work, we designed *INAM* [13] to understand the network communication pattern of InfiniBand. However, it still lacks the ability to profile and visualize the MPI communication pattern in a network topology-aware manner. As IB clusters grow in size and scale, it becomes critical for the users of HPC installations to clearly understand how an HPC application interacts with the underlying IB network and the impact it can have on the performance of the application.

No tool currently exists that allows users of such large scale clusters to analyze and to visualize the communication pattern of their MPI based HPC applications in a network topology-aware manner. Most contemporary MPI profiling tools for IB clusters also have an overhead attached to them. This prevents users from deploying such profilers on large-scale production runs intent on achieving the best possible performance from their codes. These lead us to the following broad challenge - *Can a network topology-aware, scalable, low-overhead profiler be designed for IB clusters that is capable of depicting the communication pattern of high performance MPI applications?*

In this paper we take up this challenge and design *INTAP-MPI* - a network topology-aware, scalable, low-overhead MPI profiler that allows users to analyze and visualize the communication pattern of MPI based HPC applications. INTAP-MPI gives the flexibility to profile the MPI application for either the entire duration of application execution (by means of environment variables) or for specific sections of the application (by means of Unix signals). We integrate INTAP-MPI into the MVAPICH2 MPI library [2], allowing users to seamlessly use it for analyzing their applications. Our experimental analysis shows that INTAP-MPI is able to profile and visualize the communication pattern of applications with very low memory and performance overhead at scale.

We organize the remainder of this paper as follows. We present the broad vision and possible use cases for INTAP-MPI in Section 2. Section 3 gives a brief overview of InfiniBand, IB network topology discovery and MPI over IB. In Section 4 we present the framework and design of INTAP-MPI. We evaluate and analyze the correctness and performance of INTAP-MPI in various scenarios in Section 5, describe the currently available related tools in Section 6, and summarize the conclusions and possible future work in Section 7.

2 Vision

As seen in Section 1, it is critical for users of HPC installations to clearly understand the impact IB network topology can have on the performance of HPC applications. Figure 1 illustrates how INTAP-MPI can be utilized by different users of a supercomputing installation. For instance, an MPI developer, can utilize INTAP-MPI to analyze how various MPI level collective algorithms are impacted by network topology and devise new and more network topology-aware algorithms that might prove more tolerant to network congestion. A scheduler developer / system administrator might attempt to classify the applications based on their communication pattern and make modifications to the job launching infrastructure so as to place the various ranks in a network topology-aware fashion. INTAP-MPI can also be used by scientists / application developers to re-write their applications with an eye on the underlying network topology.

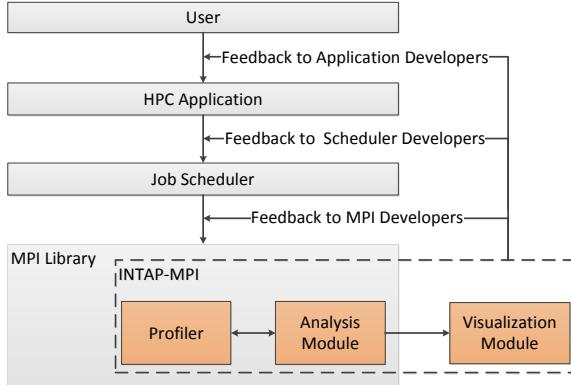


Fig. 1. Envisioned use cases for INTAP-MPI

3 Background

3.1 InfiniBand

InfiniBand is a very popular switched interconnect standard being used by almost 41% of the Top500 Supercomputing systems [18]. InfiniBand Architecture (IBA) [10] defines a switched network fabric for interconnecting processing nodes and I/O nodes, using a queue-based model. InfiniBand standard does not define a specific network topology or routing algorithm and provides the users with an option to choose as per their requirements. IB also proposes link layer Virtual Lanes (VL) that allows the physical link to be split into several virtual links, each with their specific buffers and flow control mechanisms. This possibility allows the creation of virtual networks over the physical topology. However, current generation InfiniBand interfaces do not offer performance counters for different virtual lanes.

3.2 InfiniBand Network Topology Detection Service

InfiniBand network topology data is not available in a mode that is easily used by other programs. The physical connections between entities in the fabric are discoverable with the `ibnetdiscover` utility from the OFED distribution [3]. The Logical routing data is available by querying each switch in turn and dumping its Linear Forwarding Table (LFT) with the `ibroute` utility from OFED. We leverage the basic topology detection service proposed in [8], but have since extended the method to query routing information directly within the OpenSM [15] subnet manager using a plugin interface.

3.3 MPI

Message Passing Interface (MPI) [12], is one of the most popular programming models for writing parallel applications in cluster computing area. MPI libraries provide

basic communication support for a parallel computing job. In particular, several convenient point-to-point and collective communication operations are provided. High performance MPI implementations are closely tied to the underlying network dynamics and try to leverage the best communication performance on the given interconnect. In this paper, we use modified MVAPICH2 [2] based on the 1.8 release for our evaluations. However, our observations in this context are quite general and they should be applicable to other high performance MPI libraries as well.

4 Design of Network Topology-Aware Performance Analysis Tool for MPI

The overall architecture of InfiniBand Network Topology-Aware Performance Analysis Tool for MPI (INTAP-MPI) is presented in Figure 2. It consists of two major design components: (1) light weight profiling interface and (2) topology-aware analysis module. We will go into the details of each in the following sections.

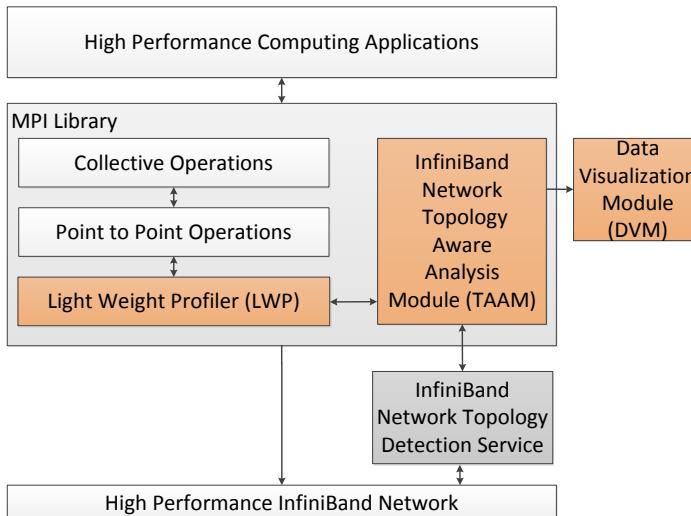


Fig. 2. Overall framework

4.1 Design of Topology-Aware Analysis Module

The Topology-Aware Analysis Module (TAAM) forms the core of INTAP-MPI. TAAM initiates and coordinates all profiling and analysis in INTAP-MPI. The user is responsible for initiating all profiling activities. This can be done either for the entire duration of application execution (by means of environment variables) or for specific sections of the application (by means of Unix signals). TAAM performs the following activities on receiving the user request to initialize the profiling: (1) informs the Light Weight

Profiling interface (LWP) (described in Section 4.2) to initiate logging the intra-node and inter-node communication inside the MPI library and, (2) queries the InfiniBand Network Topology Detection service and identifies the layout of the various processes on the InfiniBand network. Depending on the users choice (through environment variables), TAAM can either request LWP to profile either the number of messages or the volume of messages.

Once TAAM performs these activities, it remains idle until either the application terminates (calls MPI_Finalize) or receives a signal from the user requesting INTAP-MPI to finalize the profiling. On receiving the request from the user to finalize the profiling, TAAM informs the LWP to stop logging the messages. Once the LWP has stopped logging messages and handed it over to the TAAM on the local process, TAAM in *rank 0* of the MPI job gathers the communication profile from each rank. It then uses the IB network topology information obtained earlier to classify the messages logged by the LWP based on the number of hops they had to traverse. TAAM depends on the InfiniBand Network Topology Detection Service described in Section 3.2 to perform this classification. Currently, the service is only capable of distinguishing between processes based on the number of inter-node network hops. As part of future work, we plan to add capabilities to the service that will enable it to account for intra-node NUMA topologies as well. TAAM can perform this classification various granularities - process, compute node and switch blade, based on the users choice. This information is stored in a file which is later parsed by the data visualization module described in the next section.

Data Visualization Module: As the name suggests, the Data Visualization Module (DVM) visualizes the network topology-aware communication matrix generated by TAAM using standard Unix tools like *gnuplot* [1]. It generates two kinds of graphs - (1) a stacked histogram showing the splitup of physical communication based on the number of network hops and (2) a heatmap depicting the relative volume of various types of message transfers (intra-node, inter-node-1-hop, inter-node-3-hops, inter-node-5-hops, etc). Depending on the granularity of the data generated by TAAM (process, compute node or switch blade level), the graphs generated by the DVM will also depict different patterns. This information can then be used by HPC application developers, MPI library developers as well as system administrators to decide upon the best rank / task layout for a given job. DVM also provides scripts that allow users to perform post-run comparison and analysis of the communication pattern of multiple jobs. The scripts summarize the overall communication pattern of the jobs into a single graph allowing users to reason about the performance of various jobs from a network perspective.

4.2 Design of Light Weight Profiler

The Light Weight Profiler (LWP) is responsible for logging all intra-node and inter-node communication performed by the MPI library. We integrate the LWP into the lowest communication layers of the MVAPICH2 MPI library allowing it to capture the actual communication behavior, including any fragmentation done by the MPI library for load balancing purposes. On receiving the signal from TAAM to start message logging, LWP at each process allocates an array whose size is determined by the level of profiling granularity chosen by the user. If the user wants to view the communication pattern

between each pair of processes, LWP allocates an array of size $\mathcal{O}(N_{\text{processes}})$ to log the various messages issued by the process to its peers in the MPI job. If the user desires to view the communication pattern at a compute node level granularity, LWP allocates an array of size $\mathcal{O}(N_{\text{nodes}})$. LWP continues to log the messages until it receives the signal from TAAM to stop profiling. On receiving this message, it stops profiling and transfers the collected data to the TAAM on the local process which will then be gathered by the TAAM on rank 0 of the MPI job, as described in Section 4.1.

5 Experimental Results

We describe the results of the various experiments carried out for this paper in this section.

5.1 Experimental Setup

Multiple high performance computing systems were used to obtain the results for this paper:

Ranger: Ranger is comprised of 3,936 16-way SMP compute nodes providing a total of 62,976 compute cores. Each core operates at 2.3 GHz and has 32 GB of memory with an independent memory controller per socket. Ranger has two 3,456 port SDR Sun InfiniBand Datacenter switches at its core. The interconnect topology is a 7-stage, full-CLOS fat tree.

Hyperion: This is a 1,400-core testbed where each node has eight Intel Xeon (E5640) cores (Nehalem) running at 2.53 Ghz with 12 MB L3 cache. Each node has 12 GB of memory and an MT26428 QDR ConnectX-2 HCA. It has a 171-port Mellanox QDR switch, with 11 leafs, each having 16 ports. The switches form a partial FAT tree. Each node is connected to the switch using one QDR link. Although Hyperion does have additional compute cores, they use the Harpertown range of CPU's. Hence we restrict ourselves to the nodes described above.

The experiments described in Sections 5.2 and 5.3 were obtained on the Ranger supercomputing system. However, due to lack of system resources, we ran the larger scale jobs studying the scalability of INTAP-MPI described in Sections 5.4 and 5.5 on Hyperion.

In Sections 5.2 and 5.3, we analyze the performance of two types of communication patterns, collective and point-to-point respectively, using INTAP-MPI. We use Alltoall, a common collective communication pattern, to study the impact of network topology on performance of collective operations. We analyze the performance of two different runs of the MPI_Alltoall operation from a network topology-aware point of view to study and analyze the impact network topology can have on the performance of collective operations. As the NAS MG benchmark uses mostly point-to-point communications, we use that to study the impact network topology can have on such operations. The MPI_Alltoall operation as well as the NAS MG benchmark were run on the Ranger supercomputer on separate allocations of 16 compute nodes (256 processes, 16 processes per node).

In the following visualizations of communication patterns, *Red* color represents intra-node communication, *Green* color represents 1-hop communication, *Blue* color represents 3-hop communication, *Pink* color represents 5-hop communication and *Cyan* color represents 7-hop communication. In the various heat maps to follow, the intensities of these colors represent the amount of communication between the respective entities with respect to the maximum amount of communication that any two entities in the MPI job perform. The stacked histogram representation allow us to understand how the communication pattern of individual processes change over multiple runs. The heatmaps, on the other hand, help us understand which process pairs end up doing more long distance communication over the network.

5.2 Visualizing Network Characteristics of Collective Communication

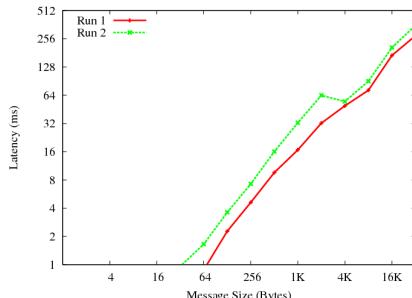
In this section, we analyze the performance of two different runs of MPI_Alltoall operation using INTAP-MPI. Figure 3(a) compares the average performance of the MPI_Alltoall operation over the two runs. Inside each run, the MPI_Alltoall operation was executed multiple times to remove possible variations in performance due to system noise. We can see that, on average, run #1 is seen to perform better than run #2. Below, we describe how an user of INTAP-MPI can use network topology-aware communication information to reason about the difference in performance of the MPI_Alltoall operation seen between the two runs.

Figure 3(b) shows the summary of the network topology-aware communication patterns of both jobs. As we can see, the number of long distance network communication (7-hops and 5-hops) is lesser in run #1 than run #2. Figures 3(c) and 3(d) depict the network topology-aware communication pattern of the MPI_Alltoall operations as stacked histograms at node level granularity. Figures 3(e) and 3(f) illustrate the same communication pattern as heat maps at node level granularity.

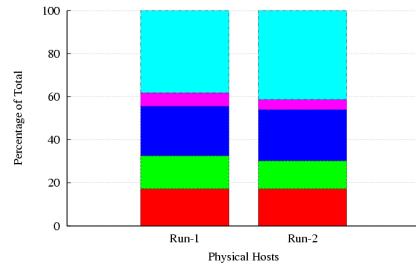
5.3 Visualizing Network Characteristics of NAS Application Benchmarks

In this section, we analyze the performance of two different runs of a class C NAS MG benchmark operation using INTAP-MPI. Figure 4(a) compares the average performance of the NAS-MG benchmark performed over the two runs. Inside each run, the NAS-MG benchmark was executed multiple times to remove possible variations in performance due to system noise. We can see that, on average, run #2 is seen to perform better than run #1. Below, we describe how an user of INTAP-MPI can use network topology-aware communication information to reason about the difference in performance of the class C NAS-MG benchmark seen between the two runs.

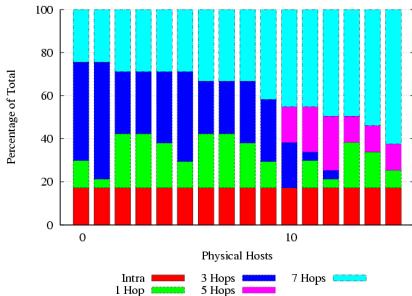
Figure 4(b) shows the summary of the network topology-aware communication patterns of both jobs. As we can see, the number of long distance network communication (7-hops) is lesser in run #2 than run #1. Figures 4(c) and 4(d) illustrate the network topology-aware communication pattern of the NAS-MG benchmark as stacked histograms at node level granularity. Figures 4(e) and 4(f) depict the same communication pattern as heat maps at node level granularity.



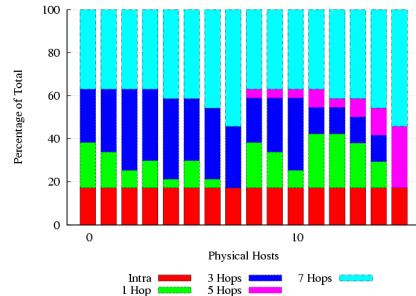
(a) Performance comparison of MPI_Alltoall between run #1 and run #2.



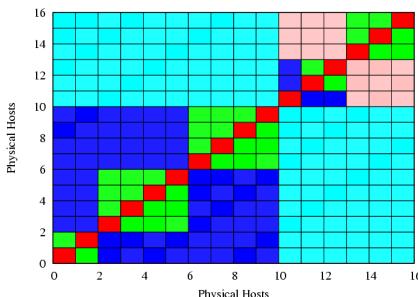
(b) Summary graph comparing communication patterns of run #1 and run #2.



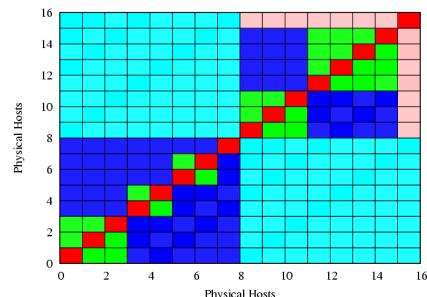
(c) Stacked histogram depicting network topology-aware communication pattern for run #1 at node level granularity.



(d) Stacked histogram depicting network topology-aware communication pattern for run #2 at node level granularity.



(e) Heatmap depicting network topology-aware communication pattern for run #1 at node level granularity.



(f) Heatmap depicting network topology-aware communication pattern for run #2 at node level granularity.

Fig. 3. Analysis of 256 process MPI_Alltoall operation on Ranger using INTAP-MPI

5.4 Impact of INTAP-MPI on Memory Consumption

Table 1 shows the impact INTAP-MPI has on the total memory consumption of the MPI job. As we saw in Section 4, only *rank 0* of the MPI job needs to allocate large memory data structures like the $\mathcal{O}(N_{\text{nodes}}^2)$ matrix required to store the communication matrix of the job at various granularities (process, node, switch blade). The memory consumption at each individual rank is only of the order of $\mathcal{O}(N_{\text{nodes}})$ bytes. As we can see, the overhead imposed by INTAP-MPI on the memory consumption of the entire application is very minimal at scale. As number of nodes reaches $\mathcal{O}(10^5)$, we can start looking at switch blade level granularity to save memory.

Table 1. Overhead (in MegaBytes) of INTAP-MPI on memory consumption of MPI_Alltoall on Hyperion

Job Size (#Processes)	64	128	256	512	1,024
Memory Overhead	0.04	0.16	0.58	2.19	8.61

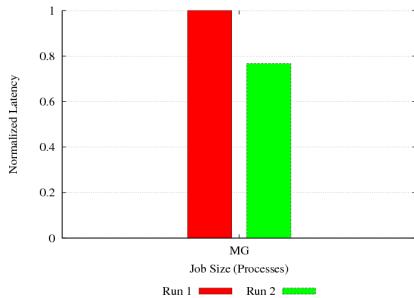
5.5 Impact of INTAP-MPI on Performance of MPI Jobs

Figure 5 depicts the performance of the MPI_Alltoall collective operation for various message sizes and system sizes. Figure 5(a) compares the communication performance of the collective operation with and without profiling at a system size of 1,024 processes. As we can see, there is very little impact on the communication performance due to INTAP-MPI. Figure 5(b) compares the performance of the MPI_Alltoall collective operation at various system sizes for a message size of 64 KB. As seen in Figure 5(a), INTAP-MPI has very little overhead on the communication performance.

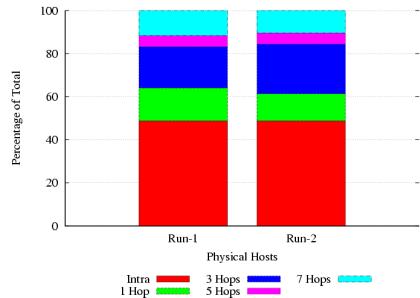
6 Related Tools

The benefit of data visualization and performance analysis tools have been demonstrated since Prism [16]. A classical MPI profiling tool, mpiP [19], is able to display details of the performance of MPI calls through MpiPView but cannot provide any topology information. Some tools like IPM [6] or Intel Trace Analyzer and Collector (ITAC), based on Vampir [14], are able to generate the communication matrix of the messages sent and received between each task, but this information is independent of the process mapping.

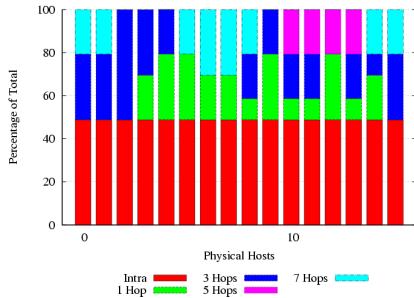
For IBM Blue Gene and Cray clusters, Bhathele [4] developed a Topology Manager API allowing an easy access to the topology information. This knowledge of the topology was possible through system calls. It allowed the development of pattern language for optimizing communication [5] or heuristics [4] included inside CHARM++ [11] to provide a topology-aware process mapping. Using similar techniques, some profiling tools, like TAU [17] or the Scalasca performance toolkit [7], are able to display the topology mapping of the processes on these systems. Nevertheless, none of the current tools is able to profile and display the mapping information of MPI processes on InfiniBand clusters in a network topology-aware manner.



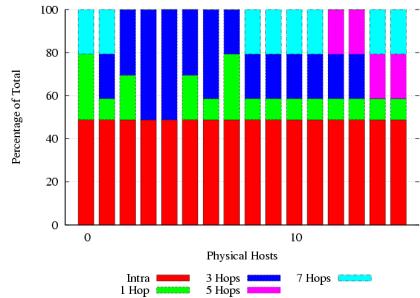
(a) Performance comparison of class C NAS MG benchmark between run #1 and run #2.



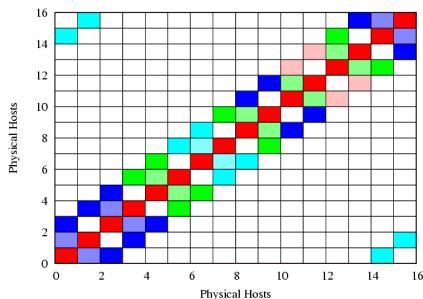
(b) Summary graph comparing communication patterns of run #1 and run #2.



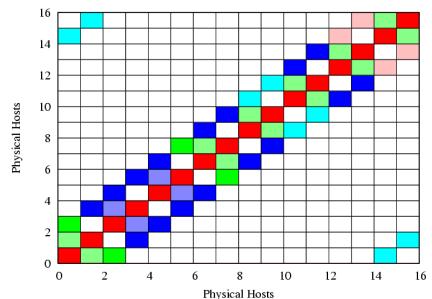
(c) Stacked histogram depicting network topology-aware communication pattern for run #1 at node level granularity.



(d) Stacked histogram depicting network topology-aware communication pattern for run #1 at node level granularity.



(e) Heatmap depicting network topology-aware communication pattern for run #1 at node level granularity.



(f) Heatmap depicting network topology-aware communication pattern for run #1 at node level granularity.

Fig. 4. Analysis of 256 process class C NAS MG benchmark on Ranger using INTAP-MPI

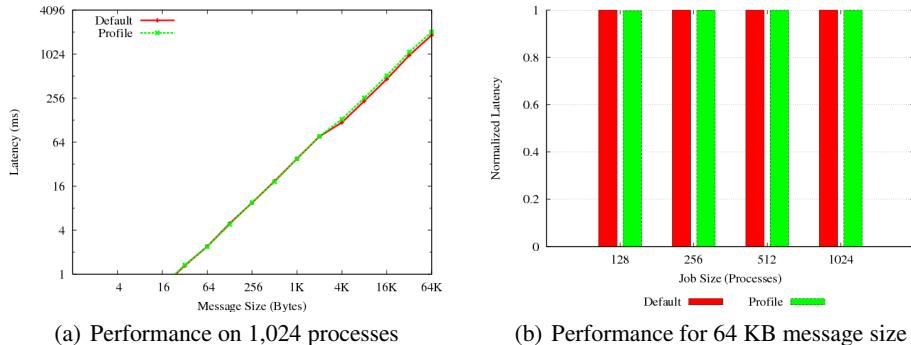


Fig. 5. Impact of INTAP-MPI on performance MPI_Alltoall on Hyperion

7 Conclusions and Future Work

In this paper, we presented the design, evaluation and use case driven analysis of INTAP-MPI - a network topology-aware, scalable, low-overhead MPI profiler that allows users to analyze and visualize the communication pattern of MPI based HPC applications for any IB network architecture (FAT Tree, Tori, or Mesh). INTAP-MPI gives the flexibility to profile the MPI application either for the entire duration of application execution (by means of environment variables) or for specific sections of the application (by means of Unix signals). We integrated INTAP-MPI into the MVAPICH2 MPI library, allowing users to seamlessly use it for analyzing their applications. Our experimental analysis showed that INTAP-MPI is able to profile and visualize the communication pattern of applications with very low memory and performance overhead at scale. In future, we would like to extend this work to make it easily usable for other MPI libraries and software stacks.

References

1. Gnuplot, <http://www.gnuplot.info/>
2. MVAPICH2: High Performance MPI over InfiniBand, 10GigE/iWARP and RoCE, <http://mvapichcse.ohio-state.edu/>
3. Open fabrics enterprise distribution, <http://www.openfabrics.org/>
4. Bhatele, A.: Automating Topology Aware Mapping for Supercomputers. Ph.D. thesis, Dept. of Computer Science, University of Illinois (August 2010)
5. Bhatele, A., Kale, L.V., Chen, N., Johnson, R.E.: A Pattern Language for Topology Aware Mapping (June 2009)
6. Fürlinger, K., Wright, N.J., Skinner, D.: Performance Analysis and Workload Characterization with IPM. In: Parallel Tools Workshop, pp. 31–38 (2009)
7. Geimer, M., Wolf, F., Wylie, B., Ábrahám, E., Becker, D., Mohr, B.: The Scalasca Performance Toolset Architecture. *Concurr. Comput.: Pract. Exper.* 22(6), 702–719 (2010)
8. Subramoni, H., Kandalla, K., Vienne, J., Sur, S., Barth, B., Tomko, K., McLay, R., Schulz, K., Panda, D.K.: Design and Evaluation of Network Topology-/Speed- Aware Broadcast Algorithms for InfiniBand Clusters. In: CLUSTER (2011)

9. Hoefler, T., Snir, M.: Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In: Proceedings of the 2011 ACM International Conference on Supercomputing, ICS 2011, pp. 75–85. ACM (June 2011)
10. InfiniBand Trade Association, <http://www.infinibandta.org/>
11. Kalé, L., Krishnan, S.: CHARM++: A Portable Concurrent Object Oriented System Based on C++. In: Proceedings of OOPSLA 1993, pp. 91–108. ACM Press (September 1993)
12. MPI Forum: MPI: A Message Passing Interface. In: Proceedings of Supercomputing (1993)
13. Dandapanthula, N., Subramoni, H., Vienne, J., Kandalla, K., Sur, S., Panda, D.K., Brightwell, R.: INAM - A Scalable InfiniBand Network Analysis and Monitoring Tool. In: Alexander, M., D’Ambra, P., Belloum, A., Bosilca, G., Cannataro, M., Danelutto, M., Di Martino, B., Gerndt, M., Jeannot, E., Namyst, R., Roman, J., Scott, S.L., Traff, J.L., Vallée, G., Weidendorfer, J. (eds.) Euro-Par 2011 Workshops, Part II. LNCS, vol. 7156, pp. 166–177. Springer, Heidelberg (2012)
14. Nagel, W.E., Arnold, A., Weber, M., Hoppe, H.C., Solchenbach, K.: VAMPIR: Visualization and Analysis of MPI Resources. *Supercomputer* 12, 69–80 (1996)
15. OFED: Open Subnet Manager,
<http://www.openfabrics.org/downloads/management/README>
16. Sistare, S., Allen, D., Bowker, R., Jourdenais, K., Simons, J., Title, R.: A Scalable Debugger for Massively Parallel Message-Passing Programs. *IEEE Parallel Distrib. Technol.* 2(2), 50–56 (1994)
17. Spear, W., Malony, A.D., Lee, C.W., Biersdorff, S., Shende, S.: An Approach to Creating Performance Visualizations in a Parallel Profile Analysis Tool. In: Alexander, M., D’Ambra, P., Belloum, A., Bosilca, G., Cannataro, M., Danelutto, M., Di Martino, B., Gerndt, M., Jeannot, E., Namyst, R., Roman, J., Scott, S.L., Traff, J.L., Vallée, G., Weidendorfer, J. (eds.) Euro-Par 2011 Workshops, Part II. LNCS, vol. 7156, pp. 156–165. Springer, Heidelberg (2012)
18. Top500: Top500 Supercomputing systems (November 2010),
<http://www.top500.org>
19. Vetter, J.S., McCracken, M.O.: Statistical Scalability Analysis of Communication Operations in Distributed Applications. In: Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, PPoPP 2001, pp. 123–132 (2001)

Performance Patterns and Hardware Metrics on Modern Multicore Processors: Best Practices for Performance Engineering

Jan Treibig, Georg Hager, and Gerhard Wellein

Erlangen Regional Computing Center (RRZE)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstr. 1, 91058 Erlangen, Germany

{jan.treibig,georg.hager,gerhard.wellein}@rrze.fau.de

Abstract. Many tools and libraries employ hardware performance monitoring (HPM) on modern processors, and using this data for performance assessment and as a starting point for code optimizations is very popular. However, such data is only useful if it is interpreted with care, and if the right metrics are chosen for the right purpose. We demonstrate the sensible use of hardware performance counters in the context of a structured performance engineering approach for applications in computational science. Typical performance patterns and their respective metric signatures are defined, and some of them are illustrated using case studies. Although these generic concepts do not depend on specific tools or environments, we restrict ourselves to modern x86-based multicore processors and use the likwid-perfctr tool under the Linux OS.

1 Introduction and Related Work

Hardware performance monitoring (HPM) is regarded as a state of the art advanced tool to guide code optimizations. While there are countless publications about HPM-based optimization efforts, a structured method for using hardware events is often missing. One exception is the use of cache miss events, which are very popular since memory access is regarded to be a major bottleneck on modern architectures. In fact, miss events are often seen as the most useful metrics in HPM. Many optimization efforts solely focus on minimizing cache miss ratios [1]. Another popular application of HPM is to measure a set of events during runtime and automatically apply optimizations online while the code is running [2,3]. The PerfExpert tool provides expert advice through automatic analysis based on HPM data of an application code [4]. Recent work attempts to apply statistical methods such as regression analysis to achieve automatic application characterization based on HPM [5,6].

This paper will present a more holistic view on how to use HPM in a sensible way. It suggests HPM as one aspect embedded in a structured performance engineering approach (see Fig. 1). The central idea is the iterative development of a diagnostic performance model enforcing a better understanding of the code properties and the hardware capabilities, leading to a deeper insight in how a code interacts with a given architecture. (A simple example of such an approach is the “roofline model” [7], which allows

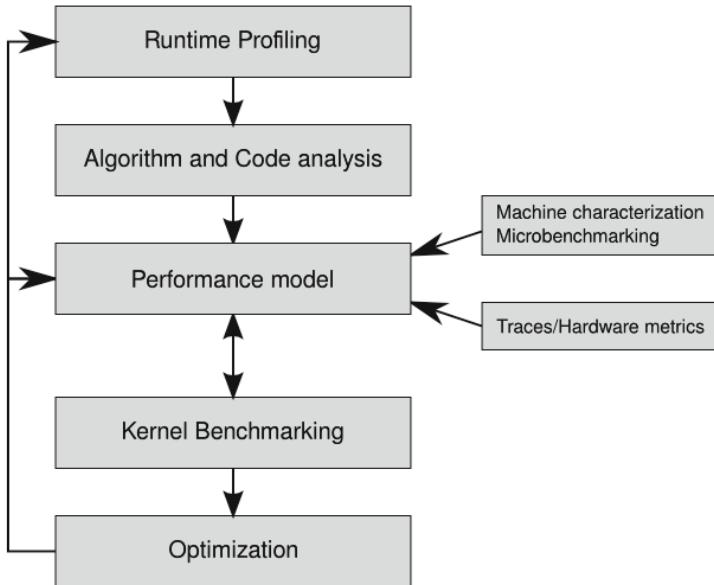


Fig. 1. Structured performance engineering process. The HPM results come in via traces and hardware metrics, but machine parameters and microbenchmarking are of equal importance. A performance model for each core loop is constructed from this data and successively refined and adapted during the benchmarking and optimization process.

to determine whether a loop's performance is limited by memory bandwidth.) HPM is one important source of information to improve this knowledge. We want to stress that HPM is in many cases only meaningful if related to other information like, e.g., microbenchmark results or static code analysis. In the following we will concentrate on what role HPM can play in order to identify performance properties and problems, and to implement a structured optimization effort.

While all these ideas are not new, we believe that the emphasis on “patterns” can help tackle problems that are not so easily recognized using automatic tools that are too focused on HPM. We concentrate here on the typical patterns and their identification. The structured performance engineering approach will be published in detail elsewhere.

1.1 Hardware Performance Metrics

Hardware performance counters are available in every modern microprocessor design. They allow the measurement of many (sometimes hundreds) of metrics that are related to the way code is executed on the hardware. Although many of those metrics are unimportant for the developer writing numerical simulation code, some of them can be very useful in assessing resource utilization and general performance properties. A large variety of tools exist, from basic to advanced, that allow easy access to HPM data, and some of them even give optimization advice derived from the measurements.

Fortunately, although there is considerable variation in the kinds of hardware events that are available on different processors (even from the same manufacturer), a rather small subset of them is sufficient to identify the most prevalent performance problems in serial and parallel code. These are available on all modern processor designs. We call a specific combination of hardware event counts and possible other sources of information a “signature.” Together with information about runtime performance behavior and code properties, signatures indicate the presence of so-called “performance patterns,” which help to assess the quality of code and, most importantly, identify relevant bottlenecks to enable a structured approach to performance optimizations.

This paper is organized as follows. In Sect. 2 we introduce a (non-exhaustive) list of performance patterns and the typical metric signatures that go with them. Sect. 3 then presents two case studies from different sectors of computational science, and Sect. 4 gives a summary and outlook to future work.

1.2 likwid-perfctr

Given sufficient experience, simple and lightweight tools are often adequate to accomplish the goals described above. Hence, we restrict ourselves to x86 architectures under the Linux OS and employ the likwid-perfctr tool from the LIKWID toolsuite [8,9]. LIKWID¹ is a collection of command line programs that facilitate performance-oriented program development and production in x86 multicore environments under Linux. The concept of event sets with connected derived metrics, which is implemented in likwid-perfctr by means of performance groups, fits well to the signature approach presented in this paper. We will not go into details on how to employ likwid-perfctr, since other tools and frameworks can do similar things.

2 Performance Patterns and Event Signatures

The following performance patterns have been found to be most useful when analyzing scientific application codes on multicore-based nodes. Other application domains may have different issues, but the basic principle could still be applied. The categorization is to some extent arbitrary, and some patterns are frequently found together.

- **Load imbalance**

Load balancing issues are an impediment for parallel scalability, and hence performance, and they should be resolved first.

- **Bandwidth saturation**

Whenever the bandwidth of a shared data path is exhausted, there is a natural limit to scalability. Most frequently this happens on the main memory interface or the (usually shared) outer-level cache (OLC).

- **Strided or erratic data access**

Cache-based architectures require contiguous data accesses to make efficient use of bandwidth due to the cache line concept. Strided access (often caused by inappropriate data structures or badly ordered loop nests) is one of the most frequent causes for low data transfer efficiency (between cache levels and to/from memory).

¹ “Like I Knew What I’m Doing”

- **Bad instruction mix**

Inefficient code execution due to an instruction mix that is inadequate to solve the problem can be a complex issue. It encompasses diverse effects such as general purpose instruction overhead created by inefficient compiler code (often occurs with C++), but also the degree of vectorization or the use of expensive operations like divide and square root.

- **Limited instruction throughput**

There is always a limit for the number of instructions that can be executed per cycle (e.g., 4 or 6), independent of their types. Even if a code does not hit this limit, it could still suffer from a bottleneck in a specific execution port (such as load or multiply). Finally, dependencies could cause pipeline bubbles, which further diminish the throughput. This pattern is closely related to the bad instruction mix pattern, but tends to require different code optimization strategies.

- **Microarchitectural anomalies**

This is a very architecture-specific pattern which may have different manifestations depending on the type of CPU. Typical examples are false store forward aliasing, unaligned data accesses or instruction code, and shortage of load/store buffers.

- **Synchronization overhead**

Barriers at the end of parallel loops or locks protecting shared resources may have a large performance impact if the workload between such synchronization points is too small. This pattern may also incur secondary effects like load imbalance or bad instruction mix (see above).

- **False cache line sharing**

Different threads accessing a cache line (and at least one of them modifying it) lead to frequent evictions and reloads, impacting performance a lot.

- **Bad page placement on ccNUMA**

All modern multi-socket servers are of ccNUMA type. Memory-bound codes must implement proper page placement in order to profit from the bandwidth advantages that ccNUMA provides. The two main problems with bad page placement are nonlocal data access and bandwidth contention, with load imbalance as a possible secondary effect.

Each of those patterns can be mapped to one or more “signatures,” which consist of a combination of performance behavior (scalability, sensitivity to problem size, etc.) and a particular pattern in raw or derived hardware metrics. While the former is often independent of the underlying architecture, the latter is very hardware-specific. Ideally a given tool should provide these event sets and derived metrics in a similar way on all supported processor architectures. likwid-perfctr tries to support this by “performance groups.” In Table 1 we give a correspondence of each performance pattern with its signatures in the performance behavior and to the relevant anomalies in hardware metrics (together with the likwid-perfctr performance group, if available). In some cases the signature also involves information from other sources such as microbenchmarks or static code analysis, since some HPM signatures may be easily misinterpreted. We deliberately do not give any general optimization hints, since optimization is only possible through a thorough code review together with a suitable performance model.

Table 1. Performance patterns and corresponding signatures for parallel code on multicore systems

Pattern	Performance behavior	Signature
		HPM (and likwid-perfctr group(s))
Load imbalance	Saturating speedup	Different count of instructions retired ² or floating point operations among cores (FLOPS_DP, FLOPS_SP)
OLC bandwidth saturation	Saturating speedup across cores in OLC group	OLC bandwidth comparable to peak bandwidth of a suitable microbenchmark (L3)
Memory bandwidth saturation	Saturating speedup across cores sharing a memory interface	Memory bandwidth comparable to peak bandwidth of a suitable microbenchmark (MEM)
Strided or erratic data access	Large discrepancy between simple bandwidth-based model and actual performance	Low bandwidth utilization despite LD/ST domination / Low cache hit ratios, frequent evicts/replacements (CACHE, DATA, MEM)
Bad instruction mix	Performance insensitive to problem size fitting into different cache levels	Large ratio of instructions retired to FP instructions if the useful work is FP / Many cycles per instruction (CPI) if the problem is large-latency arithmetic / Scalar instructions dominating in data-parallel loops (FLOPS_DP, FLOPS_SP, CPI is always measured)
Limited instruction throughput	Large discrepancy between actual performance and simple predictions based on max Flop/s or LD/ST throughput	Low CPI near theoretical limit if instruction throughput is the problem / Static code analysis predicting large pressure on single execution port / High CPI due to bad pipelining (FLOPS_DP, FLOPS_SP, DATA, CPI is always reported)
Microarchitectural anomalies	Large discrepancy between actual performance and performance model	Relevant events are very hardware-specific, e.g., stalls due to 4k memory aliasing, conflict misses, unaligned vs. aligned LD/ST, requeue events. Code review required, with architectural features in mind.
Synchronization overhead	Speedup going down as more cores are added / No speedup with small problem sizes / Cores busy but low FP performance	Large non-FP instruction count ² (growing with number of cores used) / Low CPI (FLOPS_DP, FLOPS_SP, CPI always measured)
False cache line sharing	Very low speedup or slowdown even with small core counts	Frequent (remote) evicts (CACHE)
Bad ccNUMA page placement	Bad/no scaling across locality domains	Unbalanced bandwidth on memory interfaces / High remote traffic (MEM)

3 Case Studies

3.1 Abstraction Penalties in C++ Code

The basis for this case study is a recent analysis of Expression Template (ET) frameworks for basic linear algebra operations [10,11]. While classic ETs show good performance for simple BLAS1-type (vector-vector) loop kernels, they have severe problems with BLAS2- and BLAS3-type operations and sparse arithmetic, since they are based on accesses to individual elements of data structures and have no notion of standard optimizations for nested loops. “Smart Expression Templates” (SETs) ameliorate this problem since they provide a high-level approach to complex loop nests and can substitute the whole operations by calls to optimized libraries (such as Intel MKL) or well-written plain C or compiler intrinsics code. (S)ET approaches must also be compared to standard coding techniques like operator overloading (which is plagued by the generation of temporary objects) and classic C loop nests.

Convoluted code like the one generated by strongly abstract C++ source when dealing with matrix-type operands typically shows the “**bad instruction mix**” pattern, since a lot of instructions are generated that are not actually needed to solve the problem. In the Expression Template example this shows most prominently in the number of retired instructions. Table 2 shows events and derived metrics for a 5000×5000 matrix-matrix multiplication using four different code versions: The “Classic” code uses traditional overloading of `operator*`() so that an expression like `C=A*B`, with A, B, and C being objects of some matrix class, results in a call to a function that implements a naive version of the matrix multiply and returns the result as a temporary copy. “Boost *uBLAS*” supports matrix operations directly with a slightly different syntax, and avoids the temporary (which is the main reason for using ETs in the first place). *Eigen3* is an SET framework that is able to recognize arithmetic expressions involving complex data types and employs an optimized version of the operation. However, it still relies heavily on the inlining capabilities of the compiler. “MKL `dgemm`” denotes a direct call to the vendor-optimized BLAS library for Intel processors.

The results show a striking agreement in the number of generated instructions between the Classic and *uBLAS* versions, although the performance of the Classic code is a factor of eight higher. In both cases the compiler has generated bloated, scalar machine code; the reason is that the access to individual matrix elements is strongly abstracted. The “Classic” code, uses an overloaded `operator(int,int)` for accessing the matrix elements in the loop nest, and the *uBLAS* relies on a similar mechanism. Both strategies impede the compiler’s view on what the actual operation is and limit its optimization capabilities. The result is a factor of five to six in retired instructions compared to *Eigen3* and MKL, of which a factor of two can be attributed to scalar (as opposed to SIMD-parallel) instruction code.

The fact that *uBLAS* is so much slower than the Classic version results from a very unfortunate loop ordering, leading to stride-5000 accesses to one of the matrices in the product (for details see [10]). As a consequence, the code becomes latency-dominated and makes inefficient use of the memory bandwidth (second column in Table 2).

² Load imbalance and frequent synchronization often go together, leading to large non-FP instruction counts that are caused by spin-waiting loops.

Table 2. Hardware counter performance analysis of the single-threaded multiplication of two large dense matrices ($N = 5000$). The given STREAM bandwidth is the practical limit for one thread on the used processor. (Adapted from [10])

	Memory Bandwidth [MByte/s]	Total Retired Instructions [10^{11}]	Cycles Per Instruction (CPI)	Performance [MFlop/s]
STREAM	11814	—	—	—
Classic	5314	12.5420	0.440861	1249
Boost <i>uBLAS</i>	630	10.1207	4.61834	156
<i>Eigen3</i>	371	2.1014	0.41168	8555
MKL dgemm	531	2.03448	0.321115	11261

This is the “**strided data access**” pattern at work. The Classic version, despite its inefficient machine code, at least implements a loop nest that has stride-one accesses to all relevant data structures. This is also reflected in the CPI metric (fourth column), which indicates massive pipeline bubbles due to long-latency loads. The Classic version can still not exhaust the available memory bandwidth for a single thread (see “STREAM”), although the kernel should be bandwidth-bound. This is again a consequence of the code spending too much time with in-core execution.

The *Eigen3* version, with the help of optimized kernels and massive inlining, achieves 76% of the MKL performance, which is impressive for compiler-generated code. The memory bandwidth of the MKL code is not so different from the *uBLAS* version, but this is purely coincidental: Memory access is not a bottleneck for the highly optimized dgemm implementation.

3.2 Medical Image Reconstruction by Backprojection

This case study was part of a work aiming at optimized volume reconstruction on multicore processors [12]. The optimization target is the open benchmark RabbitCt, which implements volume reconstruction by backprojection, which is the computational bottleneck in many medical imaging applications. From a simple roofline model analysis [7] the algorithm was identified to be bandwidth-limited on the platforms investigated. However, it turned out that a complex combination of patterns is involved here: **memory bandwidth saturation, limited instruction throughput, and load imbalance**.

In a first attempt it was checked if the memory bandwidth saturation pattern applies using the MEM performance group of likwid-perfctr. To get a meaningful bandwidth baseline a benchmark was constructed that mimics the basic data access pattern, which in this case is an array update kernel ($A(:,) = s * A(:,)$). The Intel Westmere processor used in the analysis can sustain 20.3 GB/s using all cores of one socket for this operation. Bandwidth measurements with likwid-perfctr revealed that the application showed a much lower bandwidth of roughly 10 GB/s. Hence, memory bandwidth saturation is not a limiting bottleneck for this implementation on Intel Westmere. A static code analysis (using the Intel IACA tool [13]) showed that the scattered load of necessary pixel

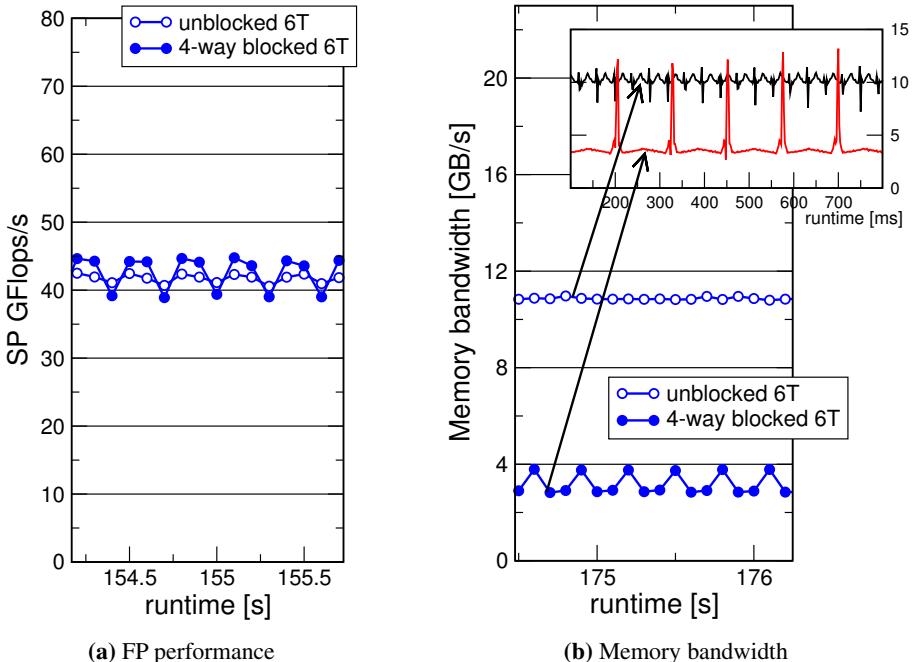


Fig. 2. Performance counter timeline monitoring of floating-point performance (a) and memory bandwidth (b), comparing blocked/nonblocked variants of the best implementation on one full Westmere socket (six cores) at 100 ms resolution. The inset in (b) shows a zoomed-in view with 2 ms resolution. (Adapted from [12])

data into SIMD registers requires a large number of instructions on the instruction code level. This makes the code limited by instruction throughput, which is not evident from the high-level language implementation. The measured performance was near the prediction of this static loop body runtime analysis, which was based on the instruction throughput capabilities of the architecture. A final confirmation that the code is indeed limited by instruction throughput was achieved by comparing with measured CPI values, which were in accordance with the static L1 cache prediction.

For a more severely bandwidth-starved architecture (Intel Harpertown) a cache-blocked version of RabbitCt was implemented, showing a significant performance improvement. This code version was also run on the Westmere platform for comparison. The result of a likwid-perfctr timeline measurement of the floating-point performance and main memory bandwidth is shown in Fig. 2: The blocking effectively lowers the bandwidth demands, but there is no impact on the overall performance.

One of the other applied optimizations was a work reduction strategy; after cutting over 30% off the total work the runtime benefit was still negligible. To check a possible load imbalance the instruction count on the cores was measured using likwid-perfctr with the FLOPS_SP group. As the innermost loop body is fully vectorized the number of packed (vectorized) arithmetic instructions is a good indicator for a potential load

Table 3. Instruction count per core for packed SSE arithmetic floating point instructions of the RabbitCt benchmark without load balancing

Core Id	0	1	2	3	4	5
FP_COMP_OPS_EXE_SSE_FP_PACKED [10^{10}]	2.74	9.39	9.23	9.30	9.29	3.07

Table 4. Instruction count per core for packed SSE arithmetic floating point instructions with round robin scheduling

Core Id	0	1	2	3	4	5
FP_COMP_OPS_EXE_SSE_FP_PACKED [10^{10}]	7.16	7.17	7.16	7.17	7.17	7.17

imbalance. Table 3 shows the results of a likwid-perfctr measurement of the packed SSE arithmetic floating point instructions. Evidently the outer threads have only one third of the workload of the others in terms of packed FP instructions. The runtime for this case is 61.72 s. A simple way to improve load balancing in OpenMP was to change the loop scheduling to a round robin distribution, using `static,1`. Results are shown in Table 4: The load imbalance was removed and the runtime was reduced to 43.9 s.

4 Summary and Outlook

This paper presented an initial proposal of a structured usage of HPM embedded in an overall software engineering process. We classify relevant performance patterns and formulate signatures which indicate that a certain pattern applies. The signatures are based on HPM data alone or combined with other sources of information such as microbenchmarking data and static code and algorithmic analysis. We are aware that this approach nevertheless requires an intimate knowledge of the algorithm, the code and the hardware. Still we believe that there is no alternative to a performance engineering process build on knowledge of the programmer himself. The application of our approach was illustrated on the example of two case studies. Future work involves further settlement of the performance patterns and corresponding signatures. This will be achieved by applying the approach to various practical examples.

References

1. Günther, F., Mehl, M., Pögl, M., Zenger, C.: A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves. *SIAM Journal on Scientific Computing* 28(5), 1634–1650 (2006). http://www5.in.tum.de/pub/int/guenther_siam06.pdf
2. Klug, T., Ott, M., Weidendorfer, J., Trinitis, C.: Autopin – automated optimization of thread-to-core pinning on multicore systems. *T. HiPEAC* 3, 219–235 (2011)
3. Chen, H., Chung Hsu, W., Lu, J., Chung Yew, P.: Dynamic trace selection using performance monitoring hardware sampling. In: *Proceedings of the 1st International Symposium on Code Generation and Optimization*, pp. 79–90 (2003)

4. Burtscher, M., Kim, B.-D., Diamond, J., McCalpin, J., Koesterke, L., Browne, J.: PerfExpert: An easy-to-use performance diagnosis tool for HPC applications. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2010, pp. 1–11. IEEE Computer Society, Washington, DC (2010), <http://dx.doi.org/10.1109/SC.2010.41>, ISBN 978-1-4244-7559-9
5. de la Cruz, R., Araya-Polo, M.: Towards a multi-level cache performance model for 3d stencil computation. Procedia Computer Science 4, 2146–2155 (2011); Proceedings of the International Conference on Computational Science, ICCS 2011, <http://www.sciencedirect.com/science/article/pii/S1877050911002936>
6. Pfeiffer, W., Wright, N.: Modeling and predicting application performance on parallel computers using HPC challenge benchmarks. In: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, pp. 1–12 (2008) ISSN 1530-2075
7. Williams, S.W., Waterman, A., Patterson, D.A.: Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Tech. Rep. UCB/EECS-2008-134, EECS Department, University of California, Berkeley (October 2008) <http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-134.html>
8. Treibig, J., Hager, G., Wellein, G.: LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. In: The First International Workshop on Parallel Software Tools and Tool Infrastructures, PSTI 2010, pp. 207–216. IEEE Computer Society, Los Alamitos (2010), <http://dx.doi.org/10.1109/ICPPW.2010.38>
9. LIKWID performance tools, <http://code.google.com/p/likwid>
10. Iglberger, K., Hager, G., Treibig, J., Rüde, U.: Expression templates revisited: A performance analysis of current ET methodologies. SIAM Journal on Scientific Computing 34(2), C42–C69 (2012), <http://dx.doi.org/10.1137/110830125>
11. Iglberger, K., Hager, G., Treibig, J., Rüde, U.: High performance smart expression template math libraries. In: Proceedings of APMM 2012, the 2nd International Workshop on New Algorithms and Programming Models for the Manycore Era at HPCS 2012, Madrid, Spain, July 2–6 (accepted, 2012)
12. Treibig, J., Hager, G., Hofmann, H.G., Hornegger, J., Wellein, G.: Pushing the limits for medical image reconstruction on recent standard multicore processors. International Journal of High Performance Computing Applications (accepted), <http://arxiv.org/abs/1104.5243>
13. Intel architecture code analyzer, <http://software.intel.com/en-us/articles/intel-architecture-code-analyzer/>

Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids

Stephen L. Scott^{1,2} and Chokchai (Box) Leangsuksun³

¹ Tennessee Tech University, TN, USA

² Oak Ridge National Laboratory, TN, USA

³ Louisiana Tech University, LA, USA

Clusters, Clouds, and Grids are three different computational paradigms with the intent or potential to support High Performance Computing (HPC). Currently, they consist of hardware, management, and usage models particular to different computational regimes, e.g., high performance cluster systems designed to support tightly coupled scientific simulation codes typically utilize high-speed interconnects and commercial cloud systems designed to support software as a service (SAS) do not. However, in order to support HPC, all must at least utilize large numbers of resources and hence effective HPC in any of these paradigms must address the issue of resiliency at large-scale.

Recent trends in HPC systems have clearly indicated that future increases in performance, in excess of those resulting from improvements in single-processor performance, will be achieved through corresponding increases in system scale, i.e., using a significantly larger component count. As the raw computational performance of these HPC systems increases from today's tera- and peta-scale to next-generation multi peta-scale capability and beyond, their number of computational, networking, and storage components will grow from the ten-to-one-hundred thousand compute nodes of today's systems to several hundreds of thousands of compute nodes and more in the foreseeable future. This substantial growth in system scale, and the resulting component count, poses a challenge for HPC system and application software with respect to fault tolerance and resilience.

Furthermore, recent experiences on extreme-scale HPC systems with non-recoverable soft errors, i.e., bit flips in memory, cache, registers, and logic added another major source of concern. The probability of such errors not only grows with system size, but also with increasing architectural vulnerability caused by employing accelerators, such as FPGAs and GPUs, and by shrinking nanometer technology. Reactive fault tolerance technologies, such as checkpoint/restart, are unable to handle high failure rates due to associated overheads, while proactive resiliency technologies, such as migration, simply fail as random soft errors can't be predicted. Moreover, soft errors may even remain undetected resulting in silent data corruption.

The goal of this workshop is to bring together experts in the area of fault tolerance and resilience for HPC to present the latest achievements and to discuss the challenges ahead.

Resilience 2012 is the follow-on workshop to the successful Resilience 2011 held in conjunction with EuroPar 2012 in Bordeaux - France, Resilience 2010 held with GGCRID 2010 in Melbourne - Australia, Resilience 2009 in Munich - Germany, and the earlier Resilience 2008 held in conjunction with CCGrid in Leon - France.

Program

The Resilience 2012 workshop program included presentations of contributed peer-reviewed papers, as well as, several invited talks.

Contributed Papers

- *"High Performance Reliable File Transfers Using Automatic Many-to-Many Parallelization."* Paul Kolano. NASA Ames Research Center (USA).
- *"A Reliability Model for Cloud Computing for High Performance Computing Applications."* Thanadech Thanakornworakij, Raja Nassar, Chokchai Leangsuksun, and Mihaela Paun. Louisiana Tech University (USA).
- *"The Viability of Using Compression to Decrease Message Log Sizes."* Kurt Ferreira, Rolf Riesen, Dorian Arnold, Dewan Ibtesham, and Ron Brightwell. Sandia National Laboratories (USA), IBM Research (Ireland), and University of New Mexico (USA).

Short Papers of Invited Talks

- *"Chaotic-identity Maps for Robustness Estimation of Exascale Computations."* Nageswara Rao. Oak Ridge National Laboratory (USA).
- *"Programming Model Extensions for Resilience at Extreme Scale."* Saurabh Hukerikar. University of Southern California (USA).
- *"User Level Failure Mitigation in MPI."* Wesley Bland. University of Tennessee (USA).

Invited Talks

- *"Does Partial Replication Pay Off?."* Dorian Arnold. University of New Mexico (USA).
- *"Resiliency: Going Forward."* Chokchai Leangsuksun. Louisiana Tech University (USA).

High Performance Reliable File Transfers Using Automatic Many-to-Many Parallelization*

Paul Z. Kolano

NASA Advanced Supercomputing Division, NASA Ames Research Center
M/S 258-6, Moffett Field, CA 94035 U.S.A.
paul.kolano@nasa.gov

Abstract. Shift is a lightweight framework for high performance local and remote file transfers that provides resiliency across a wide variety of failure scenarios. Shift supports multiple file transport protocols with automatic selection of the most appropriate mechanism between each pair of participating hosts allowing it to adapt to heterogeneous clients with differing software and network access restrictions. File system information is gathered from clients and servers to detect file system equivalence and enable path rewriting so that multiple clients can be automatically spawned in parallel to carry out both single and multi-file transfers to multiple servers selected according to load and availability. This improves both reliability and performance by eliminating single points of failure and overcoming single system bottlenecks. End-to-end integrity is provided using cryptographic hashes at the source and destination with support for partial file retransmission of only corrupted portions. This paper presents the design and implementation of Shift and details the mechanisms utilized to enhance the reliability and performance of file transfers.

1 Introduction

In high-end computing environments, remote file transfers of very large data sets to and from computational resources are commonplace as users are typically widely distributed across different organizations and must transfer in data to be processed and transfer out results for further analysis. Local transfers of this same data across file systems are also frequently performed by administrators to optimize resource utilization when new file systems come on-line or storage becomes imbalanced between existing file systems. In both cases, files must traverse many components on their journey from source to destination where there are numerous opportunities for performance optimization as well as failure. The focus of this work is to support both scenarios with an automated, high performance, high reliability tool that is simple to use and deploy.

A number of tools exist for providing reliable and/or high performance file transfer capabilities, but most either do not support local transfers, require specific security models and/or transport applications, are difficult for individual users to deploy, and/or are not fully optimized for highest performance. This paper presents Shift, which is a new framework for **Self-Healing Independent File Transfers**. Shift provides high performance and resilience for local and remote transfers through a variety of techniques.

* Supported by Task ARC-013 (Contract NNA07CA29C) with Computer Sciences Corporation

These include end-to-end integrity via cryptographic hashes, throttling of transfers to prevent resource exhaustion, balancing transfers across resources based on load and availability, and parallelization of transfers across multiple source and destination hosts for increased redundancy and performance. In addition, Shift was specifically designed to accommodate the diverse heterogeneous environments of a widespread user base with minimal assumptions about operating environments. In particular, Shift is unique in its ability to provide advanced reliability and automatic single and multi-file parallelization to any stock command-line transfer application while being easily deployed by both individual users as well as entire organizations.

Shift consists of a client and a manager component. Figure 1 shows the position of these components within a basic sequential file transfer. A single transfer may consist of many different file operations such as creating directories, copying files, changing attributes, computing checksums, etc. The original client computes the operations that comprise the given transfer and initializes them on the manager. This client, together with any others spawned dynamically, then requests a set of operations from the manager, attempts those operations, and finally reports the results back to the manager. Clients may utilize different applications to carry out file operations depending on availability, performance, and underlying system characteristics. The remainder of the paper will discuss these transfer processing steps in detail.

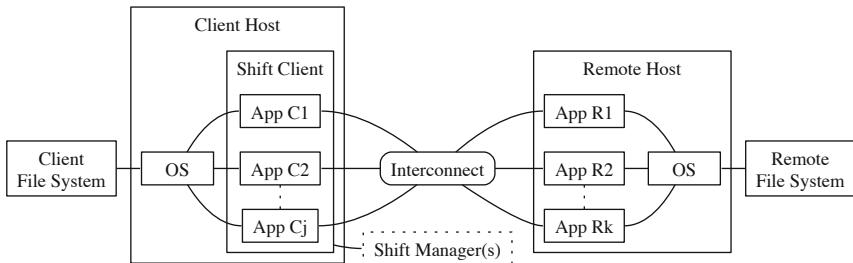


Fig. 1. Shift components within a transfer

Shift's core dependencies are Perl and direct or proxied SSH access to remote hosts using a non-interactive authentication type such as SSH public key authentication. Client hosts are not required to provide SSH access, but Shift's client parallelization features are not possible without it. Shift supports both manually-configured SSH key pairs as well as single sign-on SSH frameworks. The client has initially been tested with a lightweight authentication and authorization framework called Mesh [7], which provides single sign-on using standard SSH key pairs. With minor changes to temporary credential handling, the client can support other authentication frameworks such as the Grid Security Infrastructure (GSI) through GSI-OpenSSH [6].

This paper is organized as follows. Section 2 presents related work. Sections 3 and 4 describe the Shift manager and client. Section 5 details transfer parallelization and load balancing. Finally, section 6 presents conclusions and future work.

2 Related Work

The cp and scp utilities are the de facto standards for local and remote file transfer, respectively. A number of other file transports exist, however, that provide greater performance and/or reliability. BbFTP [3] is a remote transfer utility that supports multiple TCP streams and configurable buffer/window sizes for improved performance as well as a simple retry mechanism for improved reliability. Rsync [18] supports both local and remote transfers and can synchronize files that exist at both the source and destination using partial transfers. This increases performance by minimizing data transfer and improves reliability by correcting corruption. GridFTP [1] offers many of the features of BbFTP and Rsync with a more configurable retry mechanism and additional performance enhancements including UDP-based data streams, partial transfers, and striped transfers across multiple servers. Mcp [8] is a high performance local copy utility that supports multi-threaded single and multi-file copies, processing across multiple nodes, double buffering, and integrated parallel hashing. Shift can utilize any of these tools to take advantage of their enhancements when available.

Several projects modify existing transports to provide enhanced performance and/or reliability. Lim et al. [13] use NaradaBrokering as a more reliable communication medium between client and server within GridFTP. Sultana et al. [20] provide a technique for detecting the location of corruption in files transferred via FTP using a set of signatures appended to each file. If corruption is detected, only a small subset of the file need be transferred again (a feature Shift also provides). The need to modify both client and server software within these projects, however, makes them more difficult to deploy. HPN-SSH [16] is a performance enhancement to OpenSSH that achieves dramatic performance improvements using dynamically adjusted SSH receive windows and a multi-threaded implementation of the AES-CTR cipher. While HPN-SSH requires client and server modification to realize peak performance, either side may be modified without affecting compatibility with stock SSH installations.

Other projects (including Shift) utilize existing transports as building blocks with which to build enhanced capabilities. The Reliable File Transfer (RFT) service of the Globus Toolkit [15] adds reliability using third-party GridFTP transfers initiated from a centralized server. Since the RFT service itself becomes a single point of failure in the initial design, Basney and Duda [2] provide fault tolerance for the RFT service using multiple RFT instances with failover and a synchronized RFT transfer database. The gLite File Transfer Service (FTS) [10] is a reliable transfer service that can be layered on top of GridFTP, RFT, and other services. FTS tracks and monitors all file operations, which are carried out using third-party transfers based on lower-level services. While both RFT and FTS improve the resiliency of transfers, the use of third-party transfers will not fit into the security models of many organizations. Stork [9] is a reliable data placement framework that provides features similar to Shift including support for local transfers, automatic selection between multiple transports, and end-to-end integrity. Stork requires a long-running server accessible with GSI authentication, however, so deployment by individuals is not practical and may be difficult even for organizations.

The source file system will always be a single point of failure when only a single copy of a file exists. Replica management services such as Reptor [11] facilitate the tracking and transfer of multiple copies of the same file to provide data redundancy and

to optimize the source of a particular file. Replica Aware RFT [12] is an extension to the RFT service that allows it to utilize multiple replica servers in the transfer of a single file, thereby increasing fault-tolerance. Peer-to-peer file sharing protocols such as BitTorrent [4] offer similar functionality where clients can utilize multiple data streams for a single file to maximize network utilization from low bandwidth sources and support parallel hashing to verify the integrity of each piece. GridTorrent [21] combines the peer-to-peer functionality of BitTorrent with the speed of GridFTP to achieve high performance file sharing. Since data from individual users and/or already on local file systems is not generally replicated to other locations, Shift focuses on finding multiple access points to the same data source rather than finding multiple sources.

3 Shift Manager

Shift provides a lightweight command-line manager application that facilitates centralized tracking of file operations via two basic functions discussed throughout the remainder. *Put()* adds operations for processing or sets the state of existing operations while *get()* retrieves operations for processing. The location of the manager is shown dotted in Figure 1 as it can be deployed on any of the client host, the remote host, or a dedicated host in a redundant or standalone configuration depending on the needs of the organization or individual user. In addition to tracking, the manager also provides transfer status and performance through manual user inquiries and automated email notifications.

Shift uses a log-structured [17] flat file model for storing tracking data, which keeps dependencies on other components, such as databases, to a minimum, thereby curbing complexity and reducing points of failure. Each file operation is stored as a single line of text containing operation type (e.g. cp, mkdir, etc.), arguments, originating host, run time, size, state, and a message field. The state of a file operation is recorded in *put()* by appending the operation to the end of one of five log files corresponding to the states *do/doing/redo/done/error*. The same operation can appear in multiple logs or multiple times in the same log but the storage cost is bounded by the number of operations and configured retries possible. This model achieves the *put()* of one operation in $O(1)$ time and minimizes corruption due to outages/glitches by eliminating data overwrites.

A small metadata file for each transfer records items such as the last used position in each log, counts of operations in each state, etc., which supports the *get()* of one operation in $O(1)$ time by seeking *do/redo* to their last recorded position and returning the next file operation. Table 1 shows the cost of *put()* and *get()* for a single update of varying size. The cost of a million operations in a batch is high, but would not be used in practice since large updates reduce the effectiveness of checkpointing. *Get()* is more expensive than *put()* because it performs a number of advanced computations as described in Section 5. In a redundant manager configuration without a shared file system, tracking data must be kept synchronized. This can be achieved through standard mechanisms such as Rsync called from the manager's configurable synchronization hook. Table 2 shows the synchronization cost using Rsync for recording varying numbers of file operations via *put()* with different numbers of existing operations, which is minimal at even a million existing operations.

Table 1. Tracking cost (secs)

Op \\\ Files	1-100	1k	10k	100k	1M
put	0.094	0.10	0.20	1.3	14
get	0.15	0.19	0.56	4.8	48

Table 2. Sync. cost (secs)

Exists \\\ Puts	1	10k	100k	1M
10k	0.27	0.27	0.35	1.1
100k	0.32	0.33	0.41	1.2
1M	0.89	0.91	0.94	1.8

Table 3. Init. cost (secs)

Source Location	Path Files	/usr/bin	/usr/lib	/usr
local		0.13	0.43	2.9
LAN		0.44	3.5	24
WAN		2.7	210	-
LAN w/ helper		0.39	0.61	3.4
WAN w/ helper		5.4	11	66

4 Shift Client

The user interface to Shift’s functionality is provided by a lightweight command-line client that can function as a drop-in replacement for both cp and scp with identical usage conventions and support for the most commonly used options. This provides a known standard interface that makes usage trivial for users of Linux/Unix systems.

4.1 Initialization

A transfer begins when the client is invoked to copy files from a source to a destination. File operations are computed in a separate initialization phase that operates concurrently with file processing and allows directory traversal to complete rapidly instead of binding traversal and transfer together. Local file operations are computed using direct recursive traversal while remote operations are computed using the file manipulation features of the SFTP protocol [5]. Table 3 shows the cost of initializing a recursive transfer under various scenarios. The cost is low for local sources, but can be much higher for remote sources with the high latency of a WAN link. This cost can be greatly reduced using an optional helper script invoked over SSH to compute remote operations locally.

Operations are submitted to the manager via put(), which creates a unique identifier for the transfer, synchronizes the operations to any backup managers, if applicable, and returns the identifier back to the client for use in later operations. After initialization, the client inserts an entry into the user’s crontab that periodically invokes itself to check on the status of the transfer. If a processing thread is not detected for the transfer, the cron-invoked client begins processing the transfer itself. Hence, if a client or its associated system crashes, it will eventually be restarted to continue the transfer. The client removes the crontab when directed to do so by the manager. On systems without cron, the client parallelization discussed in Section 5 can provide similar resilience.

4.2 Multiple Transports

During initialization, a client process is forked to process batches of file operations retrieved from the manager via get() using one or more transport protocols. Shift was designed to support a variety of transports. With its basic Perl and SSH dependencies met, Shift is self-contained and can perform local copies using an integrated Perl equivalent of the cp command and remote copies using equivalents of the SFTP and FISH [14] protocols. Shift will automatically take advantage of higher performance options

when available. These currently include Mcp for local copies, BbFTP and GridFTP for remote transfers, and Rsync for both.

For each supported transport, Shift understands how to efficiently transfer a batch of files, how to detect errors, and whether errors are likely recoverable or not. Shift can support other command-line transfer applications in the same manner. The transport is selected dynamically for each batch of files in order of highest estimated performance, which will vary depending on the sizes of the files in the batch. Before any transport is used, a small test transfer is performed to ensure its availability and correct operation with the built-in options available as a last resort. Transfer performance with the various transports will be shown in Section 5.

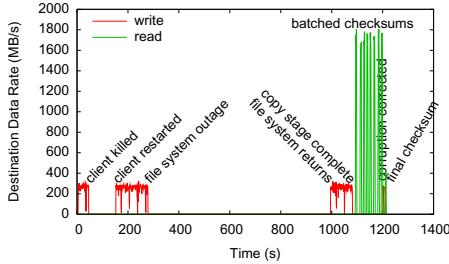
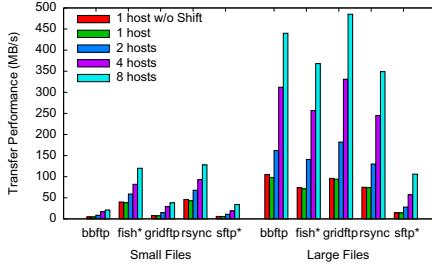
4.3 End-to-End Integrity

To detect corruption that may occur in the many components a file may traverse, Shift supports optional end-to-end integrity by computing file hashes at the source and destination after successful transmission. A matching hash value provides significant assurance that the file has arrived without corruption. If supported by the transport (e.g. Mcp and the built-in transports), Shift allows the source hash to alternatively be computed as part of the transfer itself, which can result in significant performance gains [8] as the source buffer read from disk for the transfer can be reused for the hash computation. Unlike other transfer applications that only verify bits received over the network, Shift verifies the actual bits stored on the target disk for true end-to-end verification.

Traditionally, differing hash values indicate some unknown portion of a file is corrupt. Shift instead provides a hash tree capability that indicates where the corruption is located at a configurable granularity. The embedded implementation of this capability is roughly equivalent in performance to the standard md5sum utility and is compatible with Msum [8], which is used by Shift when available as it provides significantly enhanced hashing performance. Only the portions of the file deemed corrupted are retransmitted. Partial file transfer is supported natively by Mcp and GridFTP and is also implemented in the built-in transports to augment other transports with this capability. Partially transferred files are supported in a similar manner. Namely, on an aborted transfer, Shift will determine where the source and destination differ and continue the transfer accordingly. Integrity-verified transfer performance will be shown in Section 5. Figure 2 shows a local transfer in which partial corruption and various other failures were induced to illustrate Shift’s recovery mechanisms. As can be seen, Shift is able recover from a number of different scenarios automatically.

5 Multi-host Parallelization

The systems originally chosen by the user to carry out a transfer may be non-optimal for both reliability and performance. Predetermined hosts introduce single points of failure that may not need to exist. By parallelizing a transfer across multiple equivalent hosts, the transfer may still be able to complete even if a failure has occurred along its original path. Parallelization also increases the aggregate CPUs, memory, I/O bandwidth, and network bandwidth available for the transfer, which provides increased performance when all components are functioning properly.

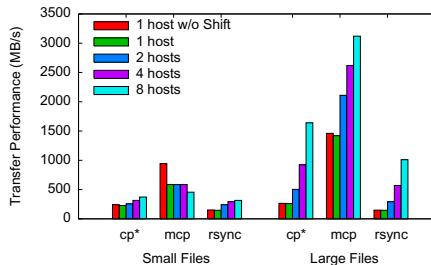
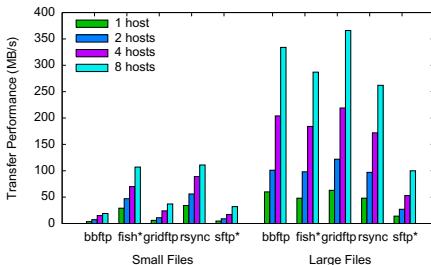
**Fig. 2.** Automated failure recovery**Fig. 3.** Remote transfer performance

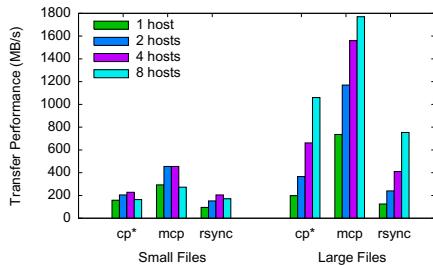
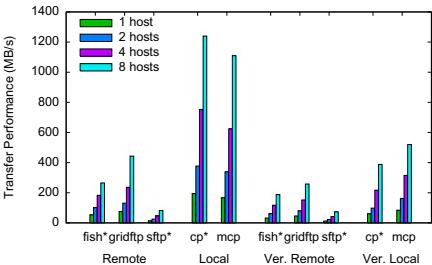
5.1 Multi-file Parallelization

Any alternate client/remote hosts used must share access to the file system utilized by the original client/remote host. Remote file system equivalence is derived from information supplied by the deploying user/organization via the periodic invocation of an included tool. The client collects similar information incrementally via the mount command during every initialization and sends it to the manager to make client parallelization decisions. In a typical cluster environment where parallelization is most effective, user logins will be distributed across a set of equivalent front-ends. Hence, over time, a complete view of the client environment will be built by simply using the Shift client.

Client parallelization occurs during get() processing. If enough work remains, the manager searches for hosts with equivalent access to the client file system based on the information transmitted during each initialization. The client is then directed to spawn itself on as many such hosts as there is work available by invoking itself on the given host(s) via SSH. All spawns are typically performed during the first get() when an SSH agent from the user's interactive session is often available for authentication if host-based authentication is not supported. Spawning clients immediately add themselves to cron and detach from the SSH session before processing operations normally via get() and put(). Since mount points may differ on each host, paths returned to clients are rewritten based on the manager's file system information. Remote hosts are also parallelized at this point by returning alternate remote paths.

Figures 3 and 4 show the performance of the various remote and local transports (built-ins denoted with '*') for a small file case of 1024 4MB files and a large file case

**Fig. 4.** Local transfer performance**Fig. 5.** Remote verified transfer performance

**Fig. 6.** Local verified transfer performance**Fig. 7.** Single file parallelization performance

of 64 1GB files without Shift on one host and with Shift on varying numbers of parallel hosts. Figures 5 and 6 show the same cases for integrity-verified transfers using Msum for hash calculations. Remote transfers were between GPFS and Lustre file systems of two cluster front-ends with 8-core 2.8/3.0 GHz Xeon Westmere/Harpertown CPUs and 1 Gb/s NICs over a 10 Gb/s WAN link with HPN-SSH installed at both sites. Local transfers were between Lustre file systems on 3.0 GHz Xeon Harpertowns.

Shift adds minimal overhead except in the local Mcp small file case where the transfer is over before the other hosts even have time to join it. Transfer parallelization achieves significantly better performance than is possible with the underlying applications on their own or the single hosts on which they normally run. This benefit was achieved automatically over the sequential case as additional host information was made available to the manager during each client initialization. Verification is CPU-intensive so can add significant overhead, but has less of an impact on slow transports where the cost of transmitting the data dwarfs the cost of hashing.

5.2 Single File Parallelization

The techniques of the previous section work best for transfers with multiple files that can be broken up into batches of roughly the equivalent size to maximize the utilization of parallel client resources. Transfers of a small number of very large files can create imbalances where some clients are stuck with the bulk of the work while the others remain idle. For this reason, Shift also supports single file parallelization where a single file can be broken up into smaller chunks that can be transferred in parallel.

This capability works by using partial file transfers. Mcp and GridFTP both have the ability to transfer a specific subset of a file beginning at a given offset with a given length. This functionality was also added to the built-in local and remote transports using standard system seek, read, and write calls and corresponding capabilities within SFTP/FISH. Files above a given split size are broken up by the manager into smaller sized chunks that can be independently transferred using the appropriate partial transfer. Figure 7 shows the benefits of using single file parallelization on a single 64GB file. As can be seen, the single file case achieves roughly 80% of the performance of the original 64 1GB file case with the exception of Mcp. Mcp performance is significantly reduced due to the disablement of direct I/O during partials transfers, which was found to exercise an undesirable Lustre bug, but still scales well in relation to single host

performance. In general, Shift's single file parallelization capability allows client workload to always be kept in balance and fully utilized regardless of the sizes of the original files. This capability also allows efficient checkpointing of large files on a single host.

5.3 Load Balancing

While parallelism greatly increases the performance that can be obtained by a single user, the use of too much parallelism by all users can quickly lead to resource contention. Shift provides several different forms of load balancing to distribute and/or reduce the load across a site. All balancing is performed during get() processing based on global transfer activity and load information sent by clients during each manager invocation. The manager first decides if the client should sleep based on user/administrator-defined thresholds for CPU, I/O, network and/or disk utilization applied to the received load information. During parallel transfers, highly loaded clients will be throttled, thereby shifting the balance of work to clients with a lighter load.

Next, the manager decides if the client should sleep based on global load and fairness criteria. Shift's basic strategy is to allow clients to initially spawn in parallel on as many resources as requested since the credentials for doing so (e.g. an SSH agent only available during a user login) may not be available a short time after invocation. The amount of work that a client actually receives, however, is governed by the manager, which, in a centralized deployment, has information on all transfers and the load they are generating on each host based on client information. If the total load is more than the site can handle, enough clients will be directed to sleep until the load is at an acceptable level. To ensure fairness at high loads, clients are throttled in round-robin fashion with each user guaranteed at least one active client before another user gets two.

Once it is determined that a client should proceed, the manager determines which remote hosts have equivalent access to the paths in the next batch of operations. The host from this set with the lowest load is then chosen and the original remote path is switched to the mount point of the selected host before being returned to the client. Using this approach, Shift allows maximum performance when resources are underutilized, but prevents degradation of performance due to overutilization.

6 Conclusions and Future Work

This paper has described Shift, a framework for **Self-Healing Independent File Transfers**. Shift uses a transfer manager to centrally track the status of all file operations throughout the life of a transfer. The manager utilizes remote file system information as well as information forwarded by clients to find hosts with equivalent access to the client and/or remote file system. If found, multiple clients may be spawned to multiple remote hosts in a many-to-many configuration. Shift adapts to many client configurations by supporting multiple file transports with automated transport selection and validation.

Shift replaces traditional sequential transfers, which are highly vulnerable to failures at every point along the path between the client and remote file systems, with a highly parallel model that is resistant to failures throughout via multiple forms of redundancy

and recovery. Outages to the client/remote file systems and the network interconnect between client/remote hosts, which are the remaining single points of failure, are tolerated through an intelligent retry mechanism that classifies failures by recoverability. Via multiple techniques, Shift provides high reliability together with high performance through aggregate resource utilization using client and manager components that are easily deployed by both organizations and individuals. Shift will be released as open source software in the near future [19].

There are a variety of directions for future work. The hash calculations for verified transfers are currently carried out sequentially where each batch of files is hashed first at the source and then verified at the destination. Ideally, all hashes would be computed in parallel and compared afterward to halve the verification cost. Support for other transports should also be investigated.

References

1. Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., Foster, I.: The Globus Striped GridFTP Framework and Server. In: ACM/IEEE Supercomputing 2005 Conf. (November 2005)
2. Basney, J., Duda, P.: Clustering the Reliable File Transfer Service. In: 2nd TeraGrid Conf. (June 2007)
3. BbFTP, <http://doc.in2p3.fr/bbftp>
4. Cohen, B.: Incentives Build Robustness in BitTorrent. In: 1st Wkshp. on Economics of Peer-to-Peer Systems (June 2003)
5. Galbraith, J., Saarenmaa, O.: SSH File Transfer Protocol. IETF Internet Draft (July 2006)
6. GSI-Enabled OpenSSH, <http://grid.ncsa.illinois.edu/ssh>
7. Kolano, P.Z.: Mesh: Secure, Lightweight Grid Middleware Using Existing SSH Infrastructure. In: 12th ACM Symp. on Access Control Models and Technologies (June 2007)
8. Kolano, P.Z., Ciotti, R.B.: High Performance Multi-Node File Copies and Checksums for Clustered File Systems. In: 24th USENIX Large Installation System Administration Conf. (November 2010)
9. Kosar, T., Livny, M.: A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems. Jour. of Parallel and Distributed Computing 65(10) (2005)
10. Kunszt, P., Badino, P., Brito da Rocha, R., Casey, J., Frohner, A., McCance, G.: The gLite File Transfer Service. In: 1st EGEE User Forum (March 2006)
11. Kunszt, P., Laure, E., Stockinger, H., Stockinger, K.: File-Based Replica Management. Future Generation Computer Systems 21(1) (January 2005)
12. Lee, Y., Kim, E., Yeom, H.Y.: Replica Aware Reliable File Transfer Service for the Data Grid. In: 4th IEEE Intl. Conf. on eScience (2008)
13. Lim, S., Fox, G., Pallickara, S., Pierce, M.: Web Service Robust GridFTP. In: 10th Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (June 2004)
14. Macheck, P.: Files transferred over SHell protocol (V 0.0.2),
http://cvs.savannah.gnu.org/viewvc/*checkout*/mc/mc/vfs/README.fish
15. Madduri, R.K., Hood, C.S., Allcock, W.E.: Reliable File Transfer in Grid Environments. In: 27th IEEE Conf. on Local Computer Networks (November 2002)

16. Rapier, C., Bennett, B.: High Speed Bulk Data Transfer Using the SSH Protocol. In: 15th ACM Mardi Gras Conf. (February 2008)
17. Rosenblum, M., Ousterhout, J.K.: The Design and Implementation of a Log-Structured File System. ACM Trans. on Computer Systems 10(1) (February 1992)
18. Rsync, <http://samba.org/rsync>
19. Shift, <http://shiftc.sourceforge.net>
20. Sultana, A., Bashir, M.F., Qadir, M.A.: CFiTT - Corrupt Free File Transfer Technique Over FTP. In: 1st IEEE Intl. Conf. on Information and Emerging Technologies (July 2007)
21. Zissimos, A., Doka, K., Chazapis, A., Koziris, N.: GridTorrent: Optimizing Data Transfers in the Grid with Collaborative Sharing. In: 11th Panhellenic Conf. on Informatics (May 2007)

A Reliability Model for Cloud Computing for High Performance Computing Applications^{*}

Thanadech Thanakornworakij¹, Raja F. Nassar¹, Chokchai Leangsuksun¹,
and Mihaela Păun^{1,2,**}

¹ College of Engineering & Science, Louisiana Tech University, Ruston, LA 71270, USA

² National Institute of Research and Development for Biological Sciences,
Bucharest, 060031, Romania
mpaun@latech.edu

Abstract. With virtualization technology, Cloud computing utilizes resources more efficiently. A physical server can deploy many virtual machines and operating systems. However, with the increase in software and hardware components, more failures are likely to occur in the system. Hence, one should understand failure behavior in the Cloud environment in order to better utilize the cloud resources. In this work, we propose a reliability model and estimate the mean time to failure and failure rate based on a system of k nodes and s virtual machines under four scenarios. Results show that if the failure of the hardware and/or the software in the system exhibits a degree of dependency, the system becomes less reliable, which means that the failure rate of the system increases and the mean time to failure decreases. Additionally, an increase in the number of nodes decreases the reliability of the system.

Keywords: Fault tolerance, Reliability, cloud computing, cloud performance.

1 Introduction

Cloud computing allows businesses to rent computing resources from cloud service providers, such as Amazon, Rackspace, etc., instead of investing to acquire computing facilities. Users can increase or decrease the amount of resources -such as storage, memory, and central processing unit (CPU) – as they need and pay per usage according to the amount of services or resources they need, like in a traditional public utility. When companies decide to use a cloud service, they have in mind a Service Level Agreement (SLA), which involves the reliability and performance of the Cloud within a desired time frame. Reliability of cloud systems directly relates to its performance. When a system fails, applications that are running on the cloud can be interrupted. If the system does not have any fault tolerant mechanisms such as live migration and checkpoint/restart, failed jobs have to re-run. If the system provides fault tolerant mechanisms there is still some performance loss; re-computing time and the

* This work was supported in part by the CNCS TE 97/2011 and NSF CNS-0834483.

** Corresponding author.

time it takes to restart the system to function again. Therefore, it is important to accurately estimate the reliability of a cloud computing system in order to better mitigate faults and therefore effectively utilize its performance to achieve the SLA of cloud users. The demand for High Performance Computing (HPC) is increasing for solving advance scientific researches and some mission critical applications. Many scientific and HPC applications are running on Cloud.

In Cloud environment, service providers must manage numerous computing components such as processors, memory modules, storage, network switches, etc. The more components, the failures are likely to occur. A failure may interrupt an entire application, for instance Message Passing Interface (MPI) applications. MPI has been used in most parallel scientific applications. MPI on Cloud is an active research area. Raihan [28] analyzed HPC applications and tested MPI performance on Cloud. If the service providers know the failure characteristic of the cloud computing components, they can better manage the computing resources to tolerate the failures and sustain better performance [7], [8].

Reliability information is one of the key factors to consider in a cloud computing environment. As such, many studies considered reliability as a main factor in their research on performance and usage cost. The work of Artur [5] focused on monetary versus reliability balance based on Spot Instances in the Amazon Elastic Computing Cloud (EC2). The Spot Instances are idle resources. Users can bid a price for idle resources. Whenever the price of resources is equal or less than the bid price, the Spot instances are allocated to the users. On the other hand, if the price goes above the bid price, the Spot instances are deallocated without warning. This leads to the question of how to bid the Spot Instances with given SLA constraints. Prasad [6] presented resource allocation method for data processing considering not just CPU speed, memory usage, data throughput, and network speed, but also reliability. In his work, Prasad presented algorithms for partitioning data in order to gain better performance and resource allocation for optimal pricing and for meeting SLA constraints. None of the work considers reliability of Cloud for HPC applications. Our work will consider the reliability of a cloud computing application that includes both hardware and software.

In this paper, we propose a reliability model for a cloud computing system that considers software, application, Virtual Machine, Hypervisor, and hardware failures as well as correlation of failures within the software and hardware.

2 Related Work

Many researchers have studied cloud computing reliability. Kashi [1] studied characteristic of hardware failure and hardware repair in cloud computing. The result showed that when a server failed, it has more chance of failing again. Also, successive failures are fit best to an inverse curve. Moreover, Dai [3] introduced cloud service reliability. The reliability of a cloud service is considered to be the reliability of request stage failures multiplied by those of the execution stage. Hacker [4] considered hardware reliability and assumed a virtual machine per server. The proposed model was based on the Weibull distribution. However, this work did not incorporate software reliability into the model. None of the work considers reliability of Cloud for

HPC applications. In our work, we will consider the reliability of a cloud computing application that includes both hardware and software

Many studies have shown that the time to failure (TTF) of a computer system can be described by a Weibull distribution [4], [9], [11]. Hacker [9] studied the impact of reliability rates of individual components on the reliability of an HPC system. He also showed that the time between failures for an HPC system followed a Weibull distribution. Xu [11] also showed TTF's of Windows NT servers follow a Weibull distribution. Gottumukkala et. al, [12] developed the reliability model of a k-node system for HPC applications when individual TTF follows a Weibull distribution. They considered the excess life or time since the last failure of an individual node in the reliability model to gain more accurate estimation of reliability of the system based on an assumption that nodes fail independently. However, time to failure of some computer systems may not be independent [11], [13]. Xu [11] showed that there is failure dependency of Windows NT Servers across the network. Schroeder [13] presented a study of failures in an HPC system at Los Alamos National Laboratory (LANL) that nodes were correlated with regard to failure and that two or more nodes may fail at the same time. In this work, we consider the excess life as well as correlation due to possible simultaneous failures of nodes in the system.

For software reliability, there are many studies undertaken to understand the characteristics of a software failure [14], [15]. Alan [10] was interested in understanding software reliability models and their utility. He showed that a simple exponential model performs as well or better than complex models, and the simple model outperformed the other models in terms of both stability and predictive ability. Musa [14] evaluated 7 model groups on 15 sets of data. Based on the evaluation, the exponential and logarithmic models were recommended for modeling software reliability. Thirumurugan [15] studied software reliability modeling in the testing and operational phase. Failures occurred at a constant rate over time [15], [16]. The majority of software reliability models make the assumption that failures occur independently. However, evidently in [17] the software failures are not always independent. Popstojanova [18] extended the classic software reliability theory, Markov renewal modeling, in order to formulate software reliability models that considered dependent failures and time to failure following the exponential distribution.

2.1 Multivariate Exponential Model

As known from the literature [14], an exponential distribution is appropriate for modeling software reliability (Applications, VM's and Hypervisor). In this work, we will refer to Application, VM's and Hypervisors as software since they all assume an exponential time till failure distribution. Software failure may be independent or dependent. Independent software failures, for example, can occur in the case of parallel jobs, such as biological sequence and video encoding. In case of dependent failure, system configuration and operation environment may cause dependent software failures. Moreover, applications may fail at the same time, due to certain situation such as communication outages between processes. For example, blocking communication on an MPI application may cause simultaneous process failures. Another example is

simultaneous failures due to the fact that applications may have to wait on data to become available. To consider correlation between software failures, we use the Marshall-Olkin Multivariate Exponential Distribution (MOMED) [19] based on the fatal shock model shown, as follows.

$$\begin{aligned}\overline{F}_{Y_1 \dots Y_k}(y_1 \dots y_k) &= \exp\left\{-\sum_{i=1}^k \gamma_i y_i - \sum_{i < j} \gamma_{ij} \max(y_i, y_j)\right. \\ &\quad \left.- \sum_{i < j < l} \gamma_{ijl} \max(y_i, y_j, y_l) - \dots - \gamma_{12\dots k} \max(y_1, y_2, \dots, y_k)\right\}\end{aligned}\quad (1)$$

Equation (1) is the probability that the system components each survive beyond times y_1, \dots, y_k , respectively.

Suppose that a component fails after receiving a fatal shock. The occurrence of shocks is based on independent Poisson process $Z_i(t, \lambda_i)$, with $i = 1, 2, \dots, k$. In each Poisson process, $t > 0$ and lambda is the Poisson parameter. The events in the Poisson process, $Z_i(t, \lambda_i)$ are shocks to component i , the events in the process, $Z_{i,j}(t, \lambda_{ij})$ are simultaneous shocks to both components i and j , the events in the process $Z_{i,j,l}(t, \lambda_{ijl})$ are simultaneous shocks to components i, j and l and the events in the process $Z_{1,2,\dots,k}(t, \lambda_{12\dots k})$ are simultaneous shocks to all k components. Thus $\overline{F}_{Y_1 \dots Y_k}(y_1 \dots y_k)$ is a survival function of k components.

2.2 Multivariate Weibull

For node failure, we have evidence that nodes may fail simultaneously [13]. Many studies also showed that the time to failure of an individual node follows a Weibull distribution. To model nodes failing at the same time, we use the Multivariate Weibull distribution. One can derive a Multivariate Weibull model from the multivariate exponential model by a variable transformation technique used in [20]. In the survival function of the multivariate exponential Eq. (1), consider the transformation $Y_i = X_i^c$, $c > 0$, $i = 1, \dots, k$. Using the transformation of variable technique, one can obtain the survival function of the general multivariate Weibull (MVW), which is given by

$$\begin{aligned}\overline{F}_{X_1 \dots X_k}(x_1 \dots x_k) &= \exp\left\{-\sum_{i=1}^k \lambda_i x_i^c - \sum_{i < j} \lambda_{ij} \max(x_i^c, x_j^c)\right. \\ &\quad \left.- \sum_{i < j < l} \lambda_{ijl} \max(x_i^c, x_j^c, x_l^c) - \dots - \lambda_{12\dots k} \max(x_1^c, \dots, x_k^c)\right\}.\end{aligned}\quad (2)$$

In what follows we extend the model in Eq. (2) to include an excess or conditional Weibull in order to determine, in conjunction with Eq. (1), the reliability of a cloud system for different scenarios that may occur in practice.

3 TTF of a Cloud System

In this section, we consider reliability of an application that has ℓ processes (App 1–App ℓ) in a cloud system composed of s virtual machines deployed in k nodes as shown in Figure 1, where each node can have a different number of virtual machines. We make an assumption that it is an MPI application. A system fails when any one of the k nodes fails or any software component fails. In the hardware case, when any node fails, it is replaced with a new node and the system is re-started. In the software case, the VM is re-started and the system operates again.

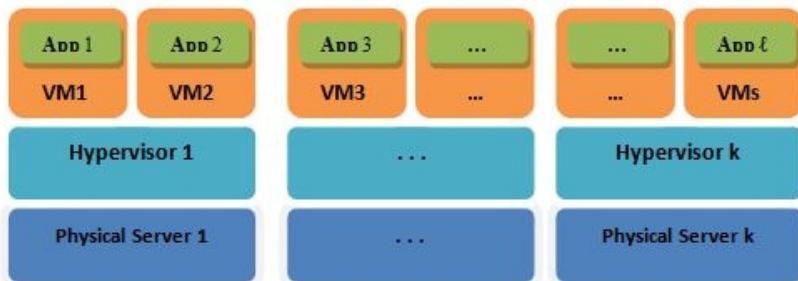


Fig. 1. Cloud Computing Architecture

Our interest is in determining the probability density function for the time to the first failure of the system after the j^{th} node replacement or a VM re-start due to a failure. This time is referred to as the time to failure (TTF). Therefore, we make the following assumptions concerning the failure properties of individual nodes in an HPC system, based on our discussion in the previous section.

1. TTF of an individual node follows a Weibull distribution
2. TTF of an application that is running on a particular VM has an exponential distribution.
3. The first failure interrupts the entire application.
4. After a failure, the node is replaced with a new node at the next time instant, and hence the system returns to operation.

We consider a k -node cloud system with n ($n = \ell + s + k$, see Figure 1) software components (App, VM and Hypervisor), each with an exponential time to failure, where any failure in any one of the n components interrupts the entire system. If any of the software components fails the component (and hence the system) will be re-started. On the other hand, if a node fails, the node is replaced and the system is restarted. In case of node failures, the distribution of the time to failure is Weibull for the node that is replaced and excess Weibull for the other nodes. Note that because of simultaneous failures; more than one node may be replaced at the same time. The excess life is defined as the probability that a node will fail at time x given that the node has survived until time t . The excess life PDF for a Weibull distribution can be expressed as

$$f(t_{ij} + x | t_{ij}) = \lambda c(t_{ij} + x)^{c-1} e^{-\lambda(-t_{ij}^c + (t_{ij} + x)^c)}, \quad (3)$$

where i is the node that fails and j is the j^{th} system re-start.

4 Cloud System Reliability

In general, there could be many possible combinations of failure dependencies among components. However, we practically focus on four major scenarios described as follows. We consider four possible scenarios.

Software failures occur independently. Also, hardware failures occur independently.

From Eqs. (1) and (2) one can show that the reliability model of n software components running on k physical nodes can be expressed as follows:

$$R_j(x) = \bar{F}(x) = P(X_1 > x, X_2 > x, \dots, X_{k+n} > x) = \exp\left\{-\sum_{i=1}^k \lambda_i x'_i - \sum_{v=1}^n \gamma_v x\right\}, \quad (4)$$

where $x'_i = x^c$ if the i^{th} node is replaced at the j^{th} re-start.

$= -t_{ij}^c + (t_{ij} + x)^c$ if the i^{th} node is not replaced at the j^{th} re-start.

Similarly, for x'_s, x'_l, \dots, x'_k . Note that for independence, only λ_i 's and γ_i 's are not equal to zero.

Eq. (5) can be readily derived from Eq. (1) by using the transformation $Y_i = X_i^c$, $c > 0$ for the nodes that have a Weibull time to failure (failed and were replaced to restart the system), $Y_i = -t_i^c + (t_i + X)^c$ for nodes that have an excess Weibull distribution (did not fail when the system was re-started) and $Y = X$ for the n software components that have an exponential time to failure.

In case of correlated software failures, and independent hardware failure, the reliability model can be expressed as in Eq. (6), which is a product of Eq. (1) and Eq. (2). Note that for independent hardware failure only λ_i 's are not equal to zero.

$$R_j(x) = \exp\left\{-\sum_{i=1}^k \lambda_i x'_i - \sum_{v=1}^n \gamma_v x - \sum_{\substack{v,w=1 \\ v < w}}^n \gamma_{vw} x - \sum_{\substack{v,w,z=1 \\ v < w < z}}^n \gamma_{vwz} x - \dots - \gamma_{12\dots n} x\right\} \quad (5)$$

In a similar manner, we derive Eq. (7) and Eq. (8) for the scenarios described below in case 3 and case 4.

For a system where software failures occur independently, but hardware failures are correlated, the reliability model is given by

$$\begin{aligned} R_j(x) = \exp\left\{-\sum_{i=1}^k \lambda_i x'_i - \sum_{\substack{i,s=1 \\ i < s}}^k \lambda_{is} \max\{x'_i, x'_s\} - \sum_{\substack{i,s,l=1 \\ i < s < l}}^k \lambda_{isl} \max\{x'_i, x'_s, x'_l\} - \dots \right. \\ \left. - \lambda_{12\dots k} \max\{x'_1, x'_2, \dots, x'_k\} - \sum_{v=1}^n \gamma_v x\right\} \end{aligned} \quad (6)$$

In the case of both software and hardware correlated failures, the reliability model of the cloud system is given by

$$\begin{aligned}
 R_j(x) = & \exp\left\{-\sum_{i=1}^k \lambda_i x_i - \sum_{\substack{i,s=1 \\ i < s}}^k \lambda_{is} \max\{x_i, x_s\} - \sum_{\substack{i,s,l=1 \\ i < s < l}}^k \lambda_{isi} \max\{x_i, x_s, x_l\} - \dots\right. \\
 & \left.- \lambda_{12\dots k} \max\{x_1, x_2, \dots, x_k\} - \sum_{v=1}^n \gamma_v x_v - \sum_{\substack{v,w=1 \\ v < w}}^n \gamma_{vw} x_v - \sum_{\substack{v,w,z=1 \\ v < w < z}}^n \gamma_{vwz} x_v - \dots - \gamma_{12\dots n} x_n\right\}
 \end{aligned} \tag{7}$$

In all four scenarios, we assume that hardware and software fail independently from each other. From Eq. (5), the pdf of an application on Cloud can be derived as

$$f(x) = [c \sum_{i=1}^k \lambda_i x''_i + \sum_{v=1}^n \gamma_v] * \exp\left\{-\sum_{i=1}^k \lambda_i x'_i - \sum_{v=1}^n \gamma_v x_v\right\}, \tag{8}$$

where $x''_i = x^{c-1}$ if the i^{th} node is replaced at the j^{th} re-start.

= $(t_{ij} + x)^{c-1}$ if the i^{th} node is not replaced at the j^{th} re-start.

From Eq. (6), the pdf for this case is

$$\begin{aligned}
 f(x) = & [c \sum_{i=1}^k \lambda_i x''_i + \sum_{v=1}^n \lambda_v + \sum_{\substack{v,w=1 \\ v < w}}^n \lambda_{vw} + \sum_{\substack{v,w,z=1 \\ v < w < z}}^n \lambda_{vwz} + \dots + \lambda_{12\dots n}] \\
 & * \exp\left\{-\sum_{i=1}^k \lambda_i x'_i - \sum_{v=1}^n \lambda_v x_v - \sum_{\substack{v,w=1 \\ v < w}}^n \lambda_{vw} x_v - \sum_{\substack{v,w,z=1 \\ v < w < z}}^n \lambda_{vwz} x_v - \dots - \lambda_{12\dots n} x_n\right\}
 \end{aligned} \tag{9}$$

After we have the reliability and pdf of an application, we will use this information to compute failure rate and mean time to failure of an application on Cloud.

5 Cloud System Reliability, Mean Time to Failure (MTTF), and Failure Rate

In this section, we show by examples, for $k = 3$ and $n=12$, the effect of survival time (T) and parameter values for joint node failures ($\lambda_{ij\dots}$) on system reliability for the four cases, Eqs. (5), (6), (7) and (8), and on the failure rate Eqs. (13), (14), (15) and (16) and MTTF, Eq. (17). For simplicity, we choose the example where there is only one joint parameter ($\gamma_{12\dots n}$). We use an example where job run length (x) equals 100 hours and the node parameters $\lambda_1, \lambda_2, \lambda_3 = 0.00005$, the 2-node joint parameters $\lambda_{12}, \lambda_{13}, \lambda_{23} = 0.00005$, the 3-node joint parameter $\lambda_{123} = 0.00003$, $C=1.5$, the 2 software-component joint parameters $\gamma_1, \dots, \gamma_{12} = 0.00003$, the 3 software-component joint parameter $\gamma_{123} = 0.0007$, and the hypervisor and VM parameters $\lambda=0.00001$. It is obvious from the reliability equations (5), (6), (7) and (8) that when the system has failures that are correlated it becomes less reliable. Moreover, when

the joint failure rates ($\lambda_{ij...}$) increase, the reliability of the system decreases. This implies that a system with independent nodes ($\lambda_{ij...}, \gamma_{ij...} = 0$) is more reliable than one where the nodes are correlated in failure. Figure 2 shows the reliability for the four scenarios. Is it seen, as expected, that scenario 1 is most reliable and scenario 4 is least reliable. For all scenario, as survival time (T) increases, the reliability decreases.

It is worth mentioning that as the number of nodes or VMs increases, the reliability of the system decreases, as seen from Eqs. (5-8).

Figure 3 shows the Cloud system MTTF. When the number of nodes increases,, mean time to failure decreases. Also, when adding more components with correlated failure, the system becomes less reliable.

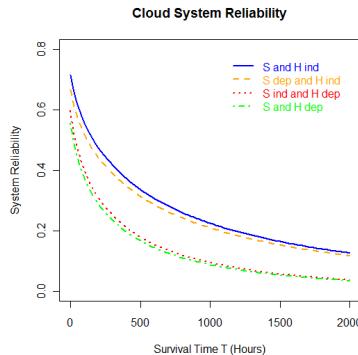


Fig. 2. Cloud System Reliability, with $k=3$ and $n=12$, for the four scenarios in Eqs. (5) – (8)

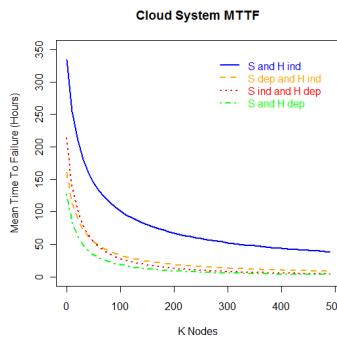


Fig. 3. Cloud System MTTF as function of the number of nodes (k)

6 Conclusions

Reliability information is important for improving Cloud computing system utilization. In order to mitigate the interruption of an application because of a failure, it is of importance to know how the reliability, failure rate, and mean time to failure of a Cloud computing system are affected by joint node failures. Many studies have shown

the existence of failure correlations among nodes in computer systems [11] and [12]. In this paper, we proposed reliability models that consider software and hardware components, which account for joint failures among nodes as well as VMs. We also developed the reliability, failure rate, and mean time to failure of a system based on k nodes and s VMs. We consider four major scenarios that are combinations of software and hardware failure correlation. We also discussed the reliability model for the case where VMs and Hypervisors exhibit an aging effect. Results showed that if failures in the system possess a degree of dependency, the system becomes less reliable. Future work will focus on extending the present models to include component redundancies.

References

1. Vishwanath, K.V., Nagappan, N.: Characterizing Cloud Computing Hardware Reliability. In: International Conference on Management of Data, pp. 193–204 (2010)
2. Yi, S., Kondo, D., Andrzejak, A.: Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In: IEEE Cloud Computing, CLOUD, pp. 236–243 (2010)
3. Dai, Y.S., Yang, B., Dongarra, J., Zhang, G.: Cloud Service Reliability: Modeling and Analysis. In: The 15th IEEE Pacific Rim International Symposium on Dependable Computing (2009)
4. Hacker, T.J.: Cloud Computing and Software Services: Theory and Techniques, pp. 139–152. CRC Press (2011)
5. Andrzejak, A., Kondo, D., Yi, S.: Decision Model for Cloud Computing under SLA Constraints. In: International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (2010)
6. Prasad, K.H., Faruque, T.A., Subramaniam, L.V., Mohania, M., Venkatachaliah, G.: Resource Allocation and SLA Determination for Large Data Processing Services over Cloud. In: Services Computing, SCC, pp. 522–529 (2010)
7. Bonvin, N., Papaioannou, T., Aberer, K.: Cost-efficient and Differentiated Data Availability Guarantees in Data Clouds. In: International Conference on Data Engineering, pp. 980–983 (2010)
8. Gueyoung, J., Joshi, K.R., Hiltunen, M.A.: Performance and Availability Aware Regeneration for Cloud Based Multitier Application. In: Dependable Systems and Networks, DSN, pp. 497–506 (2010)
9. Hacker, T.J., Meglicki, Z.: Using queue structures to improve job reliability. In: Proceedings of the 16th International Symposium on High-Performance Distributed Computing, HPDC-16 2007, Monterey, CA, pp. 43–54 (2007)
10. Wood, A.: Software Reliability Growth Models: Assumptions vs. Reality. In: The Proceedings of the Eighth International Symposium on Software Reliability Engineering, ISSRE 1997, pp. 136–141 (1997)
11. Xu, J., Kalbarczyk, Z., Iyer, R.K.: Networked Windows NT system field failure data analysis. In: Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing, pp. 178–185 (1999)
12. Gottumukkala, N.R., Nassar, R., Paun, M., Leangsuksun, C.B., Scott, S.L.: Reliability of a System of k Nodes for High Performance Computing Applications. IEEE Transactions on Reliability 59(1), 162–169 (2010)

13. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: Proceedings of International Symposium on Dependable Systems and Networks, DSN, pp. 249–258. IEEE Computer Society (2006)
14. Musa, J.D.: Software Reliability Engineering. Osborne/McGraw-Hill (1998)
15. Thirumurugan, S., Prince Williams, D.R.: Analysis of Testing and Operational Software Reliability in SRGM based on NHPP. International Journal of Computer and Information Engineering 1(1), 284–289 (2007)
16. Yang, B., Xie, M.: A study of operational and testing reliability in software reliability analysis. Reliability Engineering & System Safety Journal 70(3), 323–329 (2000)
17. Hamlet, D.: Are we testing for true reliability? IEEE Software 9(4), 21–27 (1992)
18. Goseva-Popstojanova, K., Trivedi, K.S.: Failure Correlation in Software Reliability Models. IEEE Transactions on Reliability 49(1), 37–48 (2000)
19. Marshall, A.W., Olkin, I.: A multivariate exponential distribution. Journal of the American Statistical Association 62, 30–44 (1967)
20. Hanagal, D.D.: A multivariate Weibull distribution. Economic Quality Control 11, 193–200 (1996)

The Viability of Using Compression to Decrease Message Log Sizes

Kurt B. Ferreira¹, Rolf Riesen², Dorian Arnold³, Dewan Ibtesham³,
and Ron Brightwell¹

¹ Sandia National Laboratories*, Albuquerque, NM

{kbferre,rbbriigh}@sandia.gov

² IBM Research**, Ireland

rolf.riesen@ie.ibm.com

³ University of New Mexico, Albuquerque, NM, USA

{darnold,dewan}@cs.unm.edu

Abstract. Fault-tolerance and its associated overheads are of great concern for current and future extreme-scale systems. The dominant mechanism used today, coordinated checkpoint/restart, places great demands on the I/O system and the method requires frequent synchronization. Uncoordinated checkpointing with message logging addresses many of these limitations at the cost of increasing the storage needed to hold message logs. These storage requirements are critical to the scalability of extreme-scale systems. In this paper, we investigate the viability of using standard compression algorithms to reduce message log sizes for a number of key high-performance computing workloads. Using these workloads we show that, while not be a universal solution for all applications, compression has the potential to significantly reduce message log sizes for a great number of important workloads.

1 Introduction

The reliability of future extreme-scale systems is of great concern to the high-performance computing (HPC) community. With these systems continuing to dramatically grow in size and complexity, the largest machines are becoming less reliable. In fact, failures are predicted to go from the current state of several failures per day [1, 2] to multiple failures per hour [3].

Fault-tolerance for HPC systems has been studied for several decades. Though a number of techniques have been suggested and studied, *checkpoint/restart* is

* Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

** This research was partially supported by an Enterprise Partnership Scheme grant co-funded by IBM, the Irish Research Council for Science, Engineering & Technology (IRCSET), and the Industrial Development Agency (IDA) Ireland.

the most commonly used on today's systems. During normal operation, checkpoint/restart (or *rollback recovery*) protocols [4], periodically record the state and address space of all application processes to stable storage devices. When a process fails, a new incarnation of the failed process is *recovered* from the most recent checkpoint – therefore limiting lost work.

The prevalence of checkpoint/restart is due to a number of factors: failures are a relatively rare events, applications are generally self-synchronizing, and application state can be saved and restored much more quickly than a given system's mean time to interrupt (MTTI). All of these factors have kept the overheads of traditional checkpoint/restart on current systems limited to a modest portion (currently perhaps 10-25%) of an applications total time to solution. For future extreme-scale systems, a number of these assumptions will change such that the overheads of checkpoint/restart will be prohibitively expensive [5, 6, 2], limiting application progress.

There has been a great effort in the community to optimize rollback/recovery protocols. One such optimization is uncoordinated checkpointing. Uncoordinated checkpointing allows each process to checkpoint independently, thereby avoiding synchronization overheads and reducing I/O contentions. Uncoordinated checkpointing protocols also do not require non-failed nodes to be rolled back.

As each node checkpoints independently, uncoordinated checkpointing cannot guarantee that two checkpoints from separate processes are mutually consistent and can be used together. This means that the recovery process may not be able to use the most recent set of checkpoints and ultimately can suffer the domino effect, a cascade effect that forces all nodes to rollback to the beginning of the computation. To protect against this domino effect, uncoordinated checkpointing implementations typically use sender-based message logging protocols to ensure all checkpoints taken are consistent. These logs store all message payload and envelope information in case a restarted node needs messages to be replayed.

The size of these message logs are key to scalability of HPC applications on large scale systems. Generally, writing and retrieving logs written to off-node stable storage is expensive. This is especially a concern for pessimistic logging protocols [7, 8], in which a task must be blocked until a log of its sent messages is committed to stable storage. If these logs are kept in memory [9, 4], accesses are not as expensive, but these logs occupy valuable memory on a node that could be used for the application. Further exacerbating the problem, due to power concerns, future extreme scale systems will have considerably less memory per core than current systems.

The growth rates of these logs can vary by application. Our testing using the logging library described in this paper demonstrates that per-process growth rates between 0.5MB/sec. and 40MB/sec. – an order of magnitude greater than the typical per-process, coordinated checkpoint file growth rate for these applications. If indispensable (still potentially needed) message logs grow larger than their defined space, an expensive coordination operation must be performed to ensure all nodes have successfully taken local checkpoints after which all logs can be discarded.

In this work, we investigate the viability of using standard compression techniques to reduce the size of sender-based message logs for a number of HPC workloads. Using our sender-based message-logging, uncoordinated checkpointing library we show the compression factors measured compressing the full logs versus compressing per-destination logs on a node. Per-destination logs have the possibility of improving recovery performance if multiple send operations need to be replayed. Lastly, we show the amount of message log buffering needed to achieve the best compression. The remainder of this paper is organized as follows. In Section 2 we briefly summarize the current state of the field for rollback/recovery techniques in HPC. Section 3 describes our approach used for evaluation in this paper; outlining our message-logging library, HPC workloads, and compression algorithm used. We present the result of this compression study in Section 4. Section 5 concludes the paper with a summary of our results and outlines continuing work in this area.

2 Related Rollback/Recovery Research

Resiliency and fault-tolerance has been identified by the Department of Energy and Department of Defense as one of the key fundamental challenges of extreme-scale computing. The majority of the work in this active research area has focused mostly on improving the performance of rollback/recovery methods. As described earlier, there are two major rollback/recovery variants: coordinated and uncoordinated checkpointing. A summary of these methods and their variants are described below.

2.1 Coordinated Checkpointing

Coordinated checkpointing takes regular snapshots of the global system state [10] by coordinating process checkpoints. When a process fails, the whole application is then restarted from the last known-good snapshot. Typically, two main techniques are used to coordinate checkpoints. A blocking approach quiesces the node and network prior saving global state. This synchronization can be quite expensive. In contrast, a non-blocking technique makes use of the snapshot algorithm [10] to generate a consistent checkpoint, allowing the application to continue executing.

2.2 High-Speed Storage for Checkpoint/Restart

Checkpointing to local disk and flash memory systems has periodically been proposed to speed up checkpoint/restart systems by placing large amounts of high-speed storage near the data that must be checkpointed. Actually deploying large amounts of local non-volatile storage in an extreme-scale system is potentially very challenging. Local disk-based storage has traditionally been avoided because of the increased failures it causes, for example. Upcoming non-volatile phase change PCRAM, resistive RRAM devices, and modern NAMD and NOR

flash technologies provide high bandwidth and improved reliability, but are currently very expensive. Additionally, write durability issues may require periodically replacing all flash memory in the system, further impacting total costs.

2.3 Asynchronous Checkpointing and Message Logging

Another approach that has been suggested to improve the performance of checkpointing systems is uncoordinated or asynchronous checkpointing [11–14, 4, 7, 15, 16]. These methods typically checkpoint and restore from local storage without the synchronization used by coordinated checkpointing. To support a node restoring from a local asynchronous checkpoint, nodes in this approach keep a log of recent messages that they have sent. When a node restores from a previous checkpoint, it can then replay reception of messages using a remote nodes’ logs.

While this approach can increase checkpointing performance, logging increases the latency of messaging operations and potentially takes significant amounts of memory on a node. Also, asynchronous checkpointing approaches can result in cascading rollbacks; recent work using a property called send-determinism attempts to bound the amount of rollback that may be necessary [17] and decrease log state, but this technique also places non-trivial limits on application behavior. It is important to note that one advantage of send-determinism over the compression technique described in this paper is that send-determinism reduces log size by not logging certain messages, avoid both the storage and runtime overheads of logging. We, on the other-hand, must log all messages and then compress them, but we are also guaranteed that no cascading rollbacks exist on failure, which is not true with send-determinism. In an environment where failures are common, these cascading rollbacks can have significant impact on performance. Also closely related to the work in this paper, copy-on-write has been suggested to reduce message-logging state [18]. This technique does not require additional storage for a log entry as long as that entry has not been written to since the message has been sent. For typical HPC workloads, send buffers are periodically reused in the course of the computation and are written to at least once in-between send operations. Therefore, this method typically does not significantly reduce log sizes.

3 Approach

3.1 Overview

The message-logging compression mechanism described in this paper works as follows. While the application is running, all send operations are intercepted by the message logging library. The payload of each message is written to a staging area of predetermined size. Once that staging buffer has filled, a predefined compression algorithm compresses the staging buffer and saves this compressed buffer. Along with the saved buffer, a small amount of metadata is appended to

the compressed buffer used to track messages contained within the compressed buffer.

This work looks at two policies for aggregating these message send logs on a node. The first, compresses all message payload and envelope data in program order into one buffer. The second, uses per-destination buffers for compression. In the worst case, for an n node job, this means $n - 1$ separate buffers. For many of the applications tested, nearest neighbor communication is used, therefore the number of these per-destination buffers is kept to a constant independent of node count. It is important to note that this per-destination buffer policy attempts to optimized recovery. For a failed node, considerable time can be saved by sending the compressed logs for replay rather than uncompressed.

3.2 Implementation Details

Our initial implementation for this library is at the MPI profiling layer. We chose the profiling layer for both its quick prototyping and its portability to the majority of modern MPI libraries. A full implementation of this library would benefit from being implemented at a lower layer in order to optimize the handling of some protocol messages more efficiently, specifically, restoring from failures. This initial implementation has a number of limitations. Most importantly, collective operations are implemented in terms of point-to-point operations and are not fully optimized for the job size or platform.

3.3 Applications and Platform

To evaluate the viability of this compression technique, we present results from three key HPC workloads; LAMMPS [19, 20] and two mini-applications from the Mantevo project [21], namely miniFE and HPCCG. HPCCG is a conjugate gradient benchmark code for a 3D chimney domain that can run on an arbitrary number of processors. This code generates a 27-point finite difference matrix with a user-prescribed sub-block size on each processor. miniFE mimics the finite element generation assembly and solution for an unstructured grid problem. LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a classical molecular dynamics code developed at Sandia National Laboratories. For our experiments we use the embedded atom method (EAM) metallic solid input script which is used by the Sequoia benchmark suite.

These workloads represent a range of the computational techniques frequently run at very large scales, and are representative of key simulation workloads important to both the US DOD and DOE. Since these workloads represent different communication characteristics and compute to communication ratios, etc, we expect the compression ratios for each workload to be different.

All tests were run on a small Cray XE6 system, the same architecture as the ACES Cielo platform. Each node consists of a Dual socket AMD Opteron 6136 eight-core Magny-Cours processor @ 2.4 GHz with 32 GB of main memory and Cray's Gemini proprietary network.

Lastly, for this study, we focus on the popular compression algorithm **pbzip2** [22] which had superior compression performance (compression performance being shown to be a much better predictor of success than compression/decompression bandwidths) in a previous study related to coordinated checkpoint compression [23]. **pbzip2** is a parallel implementation of **bzip2**. **pbzip2** is multi-threaded and, therefore, can leverage multiple processing cores to improve compression latency. The input to be compressed is partitioned into multiple streams that can be compressed concurrently. For these tests, the compression was performed by a spare, dedicated, on-node CPU core.

4 Compression Results

In this section, we outline the performance of this message logging compression algorithm. In this initial study we focus only on the compression factor from this technique and not on the runtime performance. The runtime overhead is of course an important key component of the benefit of this technique. In previous work we have shown that generally these compression overheads can be amortized [24]. Similar to previous work, the metric we use is compression factor. We define this compression factor as the percentage reduction in log size, therefore larger compression factors are more beneficial. For example, a 80% compression factor means the ratio of the compressed to uncompressed log is 20%.

Figure 1 shows the compression performance for HPCCG. In the figure, the staging area is the size of the send log that we accumulate before compression. We also compare the performance of accumulating into one log (denoted Full log in the figure) or separate per-destination logs on a node (denoted DST-based log). We see that HPCCG achieves a high compression factor and therefore can use this mechanism to significantly reduce message log sizes. As might be expected, one log on a node achieves superior performance to the average of the destination based logs for HPCCG. Also, the compression performance remains relatively constant for staging buffer sizes more than 8 kilobytes. As a point of reference, this technique achieves superior compression ratios in comparison to a competitive technique, send determinism, outlined in [17] and does so in a way that requires no knowledge of the applications communication and computation pattern.

Figure 2 show the compression performance of the miniFE application. Similar to HPCCG, this technique shows great potential for reducing message log sizes for the miniFE application. In contrast to HPCCG, for staging sizes less than 512KB the per-destination logs show greater compression in comparison to the one full log. To achieve the greatest compression ratio, miniFE requires a much larger staging area in comparison to an application like HPCCG.

Lastly, Figure 3 shows the compression performance for the LAMMPS simulator. For LAMMPS, both the full and per-destination logs have similar compression performance. In comparison to both HPCCG and miniFE, compression using **pbzip2** performs much more poorly on LAMMPS message log independent of the staging area. In fact, these results suggest that a technique like send-determinism may be more effective at reducing log state than compression.

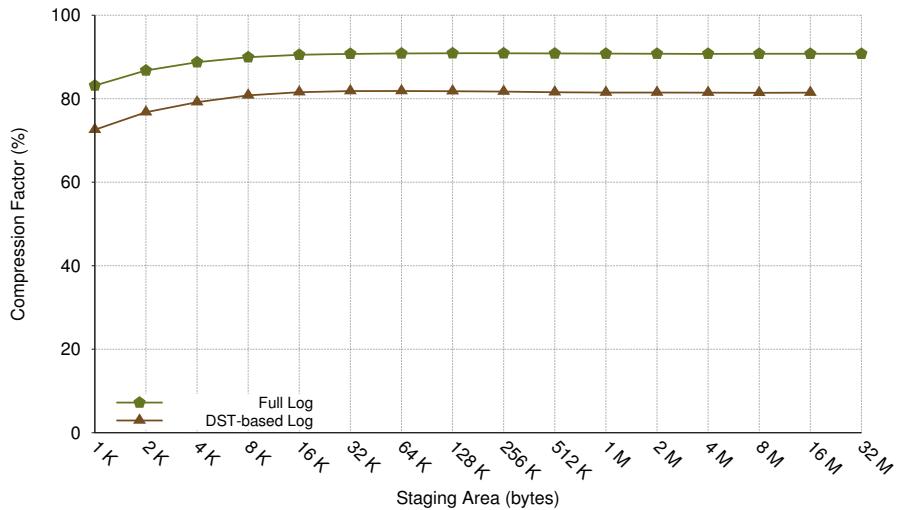


Fig. 1. Parallel bzip2 message-log compression results for HPCCG as a function of the compression staging area. Full refers to the complete message log on a node and DST is the per-destination log results.

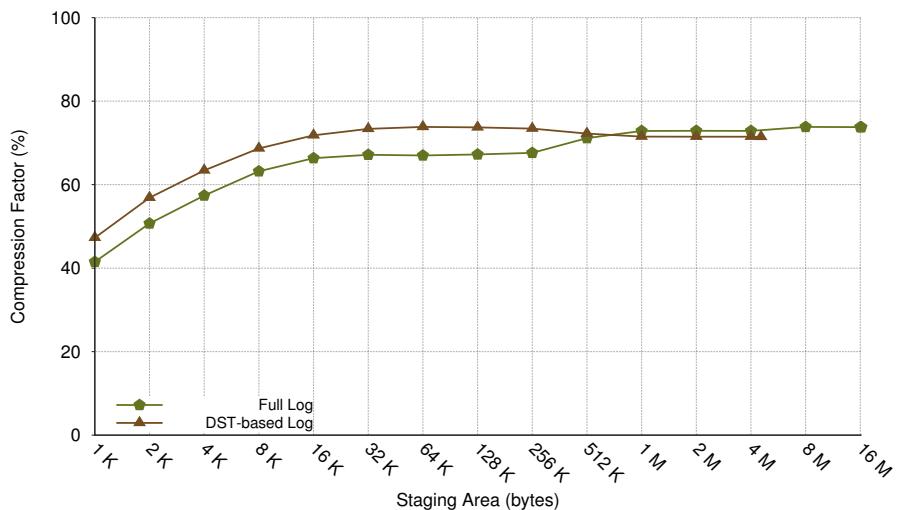


Fig. 2. Parallel bzip2 message-log compression results for miniFE as a function of the compression staging area. Full refers to the complete message log on a node and DST is the per-destination log results.

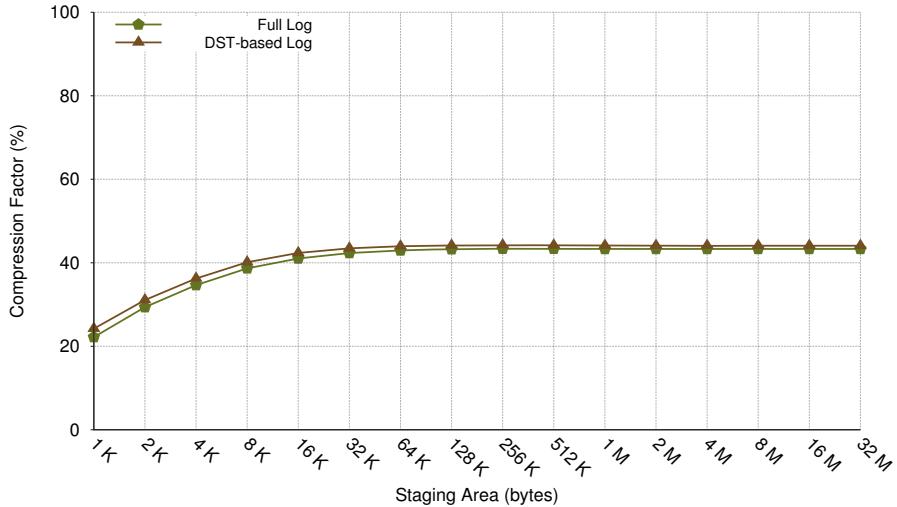


Fig. 3. Parallel bzip2 message-log compression results for LAMMPS as a function of the compression staging area. Full refers to the complete message log on a node and DST is the per-destination log results.

5 Conclusions and Continuing Work

In this paper, we investigate the viability of using standard compression techniques to reduce the size of sender-based MPI message logs, a key limiter to the scalability of message logging protocols, for a number of HPC workloads. Using a uncoordinated checkpointing with message logging library that we developed, we show that compression can significantly reduce the size of full and per-destination message logs on a node, with the method achieving greatest performance dependent on the workload. While not universally applicable to all workloads, our results show this technique can achieve compression rates far superior to competing techniques for certain classes of applications and does so without the possibility of cascading rollback and without the need of a complete understanding of the applications computation and communication patterns that other methods require. Lastly, we show that for many of these applications only a small amount of staging is needed for this method to achieve the greatest compression performance.

While this work represents an important potential viability demonstration, more work is clearly needed to fully evaluate the merits of this technique. First, in this paper we only consider the compression ratio achieved, which we have seen to be the most important metric in evaluating compression. We must also consider the associated runtime overhead of this compression mechanism and techniques to reduce those costs. Second, we are investigating studying additional HPC workloads and compression algorithms beyond those presented in this paper. Third, a more complete analysis of this technique at scale, and in comparison

with competing methods such as copy-on-write and send determinism is needed. Fourth, we plan to evaluate the impact of message log compression on reducing the latency of failure recovery and, in turn, the impact this has on application makespan under realistic failure conditions.

References

1. Schroeder, B., Gibson, G.A.: Understanding failures in petascale computers. *Journal of Physics: Conference Series* 78(1), 012022 (2007)
2. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: International Conference on Dependable Systems and Networks, DSN (June 2006)
3. Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J., Karp, S., Keckler, S., Klein, D., Kogge, P., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, R.S., Yelick, K.: Exascale computing study: Technology challenges in achieving exascale systems (September 2008), [http://www.science.energy.gov/ascr/Research/CS/DARPA_exascale-hardware\(2008\).pdf](http://www.science.energy.gov/ascr/Research/CS/DARPA_exascale-hardware(2008).pdf)
4. (Mootaz) Elnozahy, E.N., Alvisi, L., Wang, Y.-M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.* 34(3), 375–408 (2002)
5. Oldfield, R.A., Arunagiri, S., Teller, P.J., Seelam, S., Varela, M.R., Riesen, R., Roth, P.C.: Modeling the impact of checkpoints on next-generation systems. In: 24th IEEE Conference on Mass Storage Systems and Technologies, pp. 30–46 (September 2007)
6. Ferreira, K., Riesen, R., Bridges, P., Arnold, D., Stearley, J., Laros III, J.H., Oldfield, R., Pedretti, K., Brightwell, R.: Evaluating the viability of process replication reliability for exascale systems. In: Lathrop, S., Costa, J., Kramer, W. (eds.) SC. ACM (November 2011)
7. Bosilca, G., Bouteiller, A., Cappello, F., Djilali, S., Fedak, G., Germain, C., Herault, T., Lemarinier, P., Lodygensky, O., Magniette, F., Neri, V., Selikhov, A.: MPICH-V: Toward a scalable fault tolerant mpi for volatile nodes. In: Conference on High Performance Networking and Computing, SC 2002, Baltimore, MD, pp. 1–18 (November 2002)
8. Bouteiller, A., Cappello, F., Herault, T., Krawezik, G., Lemarinier, P., Magniette, F.: MPICH-V2: a fault tolerant MPI for volatile nodes based on pessimistic sender based message logging. In: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing. ACM, New York (2003)
9. Johnson, D.B., Zwaenepoel, W.: Sender-based message logging. In: Proceedings of the Seventeenth International Symposium on Fault-Tolerant Computing (1987)
10. Mani Chandy, K., Lamport, L.: Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.* 3(1), 63–75 (1985)
11. Ahn, J.: 2-step algorithm for enhancing effectiveness of sender-based message logging. In: SpringSim 2007: Proceedings of the 2007 Spring Simulation Multiconference, pp. 429–434 (2007)
12. Jiang, Q., Manivannan, D.: An optimistic checkpointing and selective message logging approach for consistent global checkpoint collection in distributed systems. In: Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium (March 2007)

13. Johnson, D.B., Zwaenepoel, W.: Recovery in distributed systems using asynchronous message logging and checkpointing. In: Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, pp. 171–181 (1988)
14. Alvisi, L., Marzullo, K.: Message logging: Pessimistic, optimistic, causal, and optimal. *IEEE Trans. Softw. Eng.* 24(2), 149–159 (1998)
15. Lemarinier, P., Bouteiller, A., Herault, T., Krawezik, G., Cappello, F.: Improved message logging versus improved coordinated checkpointing for fault tolerant MPI. In: IEEE International Conference on Cluster Computing, pp. 115–124 (2004)
16. Saito, Y., Shapiro, M.: Optimistic replication. *ACM Comput. Surv.* 37(1), 42–81 (2005)
17. Guermouche, A., Ropars, T., Brunet, E., Snir, M., Cappello, F.: Uncoordinated checkpointing without domino effect for send-deterministic MPI applications. In: International Parallel Distributed Processing Symposium, IPDPS, pp. 989–1000 (May 2011)
18. Elnozahy, E.N., Zwaenepoel, W.: On the use and implementation of message logging. In: Digest of Papers: FTCS/24, The Twenty-Fourth Annual International Symposium on Fault-Tolerant Computing, Austin, Texas, USA, June 15–17, pp. 298–307. IEEE Computer Society (1994)
19. Plimpton, S.J.: Fast parallel algorithms for short-range molecular dynamics. *Journal Computation Physics* 117, 1–19 (1995)
20. Sandia National Laboratory. LAMMPS molecular dynamics simulator (April 10, 2010), <http://lammps.sandia.gov>
21. Heroux, M.A., Doerfler, D.W., Crozier, P.S., Willenbring, J.M., Carter Edwards, H., Williams, A., Rajan, M., Keiter, E.R., Thornquist, H.K., Numrich, R.W.: Improving performance via mini-applications. Technical Report SAND2009-5574, Sandia National Laboratory (2009)
22. Elytra, J.G.: Parallel data compression with bzip2, http://gilchrist.ca/jeff/papers/Parallel_BZIP2.pdf
23. Ibtesham, D., Arnold, D., Ferreira, K.B., Bridges, P.G.: On the Viability of Checkpoint Compression for Extreme Scale Fault Tolerance. In: Alexander, M., D’Ambra, P., Belloum, A., Bosilca, G., Cannataro, M., Danelutto, M., Di Martino, B., Gerndt, M., Jeannot, E., Namyst, R., Roman, J., Scott, S.L., Traff, J.L., Vallée, G., Weidendorfer, J. (eds.) Euro-Par 2011 Workshops, Part II. LNCS, vol. 7156, pp. 302–311. Springer, Heidelberg (2012)
24. Ibtesham, D., Arnold, D., Bridges, P., Ferreira, K., Brightwell, R.: On the viability of compression for reducing the overheads of checkpoint/restart-based fault tolerance. Technical Report TR-CS-2012-01, Department of Computer Science, University of New Mexico (May 2012)

Resiliency in Exascale Systems and Computations Using Chaotic-Identity Maps

Nageswara S.V. Rao

Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
`raons@ornl.gov`

Abstract. For exascale computing systems, we propose (i) light-weight computational modules that utilize chaotic computations and customized identity maps to detect component failures, and (ii) statistical estimation methods that generate robustness estimates for the system and computations based on the module outputs. The diagnosis modules execute multiple Poincare and identity maps, which are customized to detect certain classes of failures in the compute nodes and interconnects. We propose statistical methods that generate robustness estimates for the system using the outputs of pipelined chains of diagnosis modules.

Keywords: Exascale systems, resiliency, chaotic maps, statistical estimation.

1 Introduction

Exascale computing systems are expected to have processor cores, memory units, communications and other components totaling in the numbers of millions [2]. Based on a life-span of about ten years for the components, computations of a few hours are likely to experience several component failures. As a result, outputs of codes on these systems cannot be guaranteed to be failure-free, even if failures are detected and corrected using methods such as checkpoint recovery. But such failures could have a significant impact on scientific computations. The errors could propagate through computations, such as hydrodynamic or climate codes, and potentially lead to incorrect outputs. A strategy to mitigate such effects must combine efforts to make the system robust (by methods such as checkpointing and fault-tolerant hardware), and also to enhance application codes with software fault tolerance methods.

System and application software can be designed to be robust against faults in the computing system, by utilizing combinations of methods such as checkpointing, N-version programming, process migration and more specific methods such as fault oblivious algorithms and fault tolerant MPI. However, their effectiveness drops-off exponentially with the failure levels of the system. On the other hand, it is very unproductive to design them for the worst-case failure mode, since it could degrade the computational productivity to levels of much smaller systems. Therefore, it is critical to estimate the current robustness level

of the computing system to guide these software solutions, and also to assign confidence levels to the code outputs.

From a broad perspective, methods for achieving robust computations on failure-prone computing systems have been developed in several cases. von Neumann [6] studied the mathematical aspects of building reliable computing systems from unreliable components in the 50's. These works establish analytically that it is possible to achieve robust computations even if the computing platform is inherently failure-prone. In terms of practical systems, space vehicles deployed over the past decades have been enhanced with Software-Implemented Hardware Fault Tolerance (SIHFT) methods to counteract the transient computing faults that arise due to radiation exposure. These works show that it is indeed possible to build robust computing systems capable of operating in some of the most challenging, failure-prone environments.

1.1 Our Approach

We address a class of problems of assessing the robustness levels of an exascale system at the system- and application-levels under certain failures in the compute nodes and interconnects. Our approach is based on a synthesis of methods from fault diagnosis [3] and chaotic Poincare maps [1], and consists of: (a) diagnosis methods to identify the errors due to failures using strategically guiding the execution paths, (b) Poincare maps to amplify their effects making them quickly detectable, and (c) statistical estimation methods [5] to process the data from executional traces to generate system robustness profiles and confidence estimates for application computations. The same basic methods are applied for both tasks: (i) in a strategic, pipelined configuration to estimate the system robustness, and (ii) embedded in application codes to sample their executional paths and generate confidence measures. The chaotic maps are chosen for their low computational requirements and sensitivity to computational errors arising from a wide variety of faults, including those in Arithmetic and Logic Unit (ALU), memory units and interconnects. In this extended abstract, we only briefly described our approach, and more details including preliminary proof-of-principle illustrations and simulations are provided in [4].

References

1. Alligood, K.T., Sauer, T.D., Yorke, J.A.: *Chaos: An Introduction to Dynamical Systems*. Springer-Verlag Pub., Reading (1996)
2. International exascale software project, <http://wwwexascale.org>
3. Rao, N.S.V.: Computational complexity issues in operative diagnosis of graph-based systems. *IEEE Transactions on Computers* 42(4), 447–457 (1993)
4. Rao, N.S.V.: Chaotic-Identity Maps for Robustness Estimation of Exascale Computations. In: *IEEE Workshop on Fault Tolerance for HPC at Extreme Scale* (2012)
5. Vapnik, V.N.: *Statistical Learning Theory*. John-Wiley and Sons, New York (1998)
6. von Neumann, J.: Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: Shannon, C.E., MaCarthy, J. (eds.) *Automata Studies*. Princeton University Press (1956)

Programming Model Extensions for Resilience in Extreme Scale Computing

Saurabh Hukerikar, Pedro C. Diniz, and Robert F. Lucas

Information Sciences Institute
University of Southern California
Marina del Rey CA 90292, USA
`{saurabh,pedro,rflucas}@isi.edu`

1 Introduction

The challenge of resilience is becoming increasingly important on the path to exascale capability in High Performance Computing (HPC) systems. With clock frequencies unlikely to increase as aggressively as they have in the past, future large scale HPC systems aspiring exaflop capability will need an exponential increase in the count of the ALUs and memory modules deployed in their design [Kogge 2008]. The Mean Time to Failure (MTTF) of the system however, scales inversely to the number of components in the system. Furthermore, these systems will be constructed using devices that are far less reliable than those used today, as transistor geometries shrink and the failures due to chip manufacturing variability, effects of transistor aging as well as transient soft errors will become more prevalent. Therefore the sheer scale of future exascale supercomputers, together with the shrinking VLSI geometries will conspire to make faults and failures increasingly the norm rather than the exception.

The current state of the art in techniques for designing resilient systems built from unreliable subsystems tend to employ multi-modular hardware redundancy. The overheads that these approaches incur in terms of cost and energy will be too high to be used throughout an exascale system. Software based resiliency techniques have been based on checkpoint and restart mechanisms for decades. However, checkpoint and restart will not scale to be effective when system failure rates are frequent, as are anticipated in exascale systems, since the time for checkpoint and restart will exceed the mean time to failure of the system [Cappello 2009].

System level failures attributed to memory errors are an increasing occurrence in HPC systems [Schroeder 2007]. Therefore, one of the serious challenges to system resilience is presented by the memory hierarchy with its vast size, in terms of the number of memory chips and the number of devices. Single-bit errors in memory are already common in HPC systems while multi-bit errors are becoming increasingly common [Schroeder 2009]. The increased susceptibility to multi-bit errors may be attributed to the challenges of technology scaling and the level of integration of individual devices. Most memories provide some capability for error detection, and often single bit error correction, using parity or Error Correction Code (ECC). Hardware designers have insufficient incentive to scale

the ECC approach in the future, to offer multi-bit correction because of the chip area costs and memory access latency overheads involved. Multi-bit errors which are detected but cannot be corrected by ECC logic, raise non-maskable interrupts which are delivered to the operating system which kills the application or forces a system shutdown, depending on the location of the error in memory. This in turn causes the compute node of an HPC system to crash, which terminates any processes running on it and implicitly crashes other nodes as well.

However, not all errors detected warrant catastrophic system failure. Several HPC applications are algorithmically fault tolerant, and therefore moderately tolerant to errors. But this is knowledge that the programmer has, not the system and today programmers lack convenient mechanisms to communicate their fault tolerance knowledge to the system.

2 Resilience through Programming Model Extensions

In this work, we present an approach that is based on a set of programming model extensions for system resilience that allows programmers to explicitly express their fault tolerance knowledge [Hukerikar 2012]. These resilience oriented programming model extensions are based on current and familiar language constructs and directives. These include tolerant type qualifiers for global declarations of primitive data types, tolerant code sections using preprocessor directives, a tolerant version of malloc for runtime memory allocation, a tolerant version of the looping construct and tolerant functions. Each of these allow expression of fault tolerance on specific regions of memory that are mapped to the application's active address space.

We have developed a compiler infrastructure that parses these resilience annotations and folds them into a format that may be consumed by a runtime inference engine to construct a knowledge base, called the *Dynamic Resilience Map (DRM)*. The compiler also performs certain code transformations based on the programmer annotations. The runtime engine is activated through a non-maskable interrupt raised on the occurrence of a multi-bit error. The inference engine, by referring to the *DRM*, reasons about the significance of the detected error within the address space and decides whether the error could be tolerated by the application. Therefore, a potential system crash may be prevented by leveraging the fault tolerance knowledge conveyed by the programmer.

The fault model that we use for our experimental evaluation is that of multi-bit errors that are detectable by hardware based schemes such as ECC, but not correctable in hardware. ECC schemes provide Single Error Correction and Double Error Detection (SECDED) capability and such errors in today's systems would result in a SECDED failure.

We perform a set of accelerated fault injection experiments on High Performance Computing Challenge (HPCC) Random Access benchmark. This benchmark was originally developed to model a vectorized application on a Cray Y/MP and allowed for the same address to appear twice in a gather/scatter operation and failed to retain sequential consistency. The computational kernel of RandomAccess performs memory accesses that are read-modify-write operations on

the HPCC Table. It is therefore explicitly tolerant to the presence of errors in its HPCC Table array. The benchmark is used to evaluate the peak throughput performance of the memory hierarchy as it performs random updates to the memory. The size of the HPCC Table that the benchmark operates on is allocated to occupy about half the system memory. The memory accesses are pseudo-random, i.e there is no relation between one address updated and its predecessors or successors.

In our experiments, the active address space of the application is randomly injected with multi-bit errors during execution at rates of 15, 10, 5, 2 and 1 minute respectively and for each error rate the application is executed 1000 times. The metric of success is the number of experimental runs that converge to correct solution as a fraction of all experimental runs.

Our results [Hukerikar 2012] show that even with injected error rates of 1 minute (which effectively sets the MTTF of the system to 1 minute), almost 99% of the application runs survive these errors and complete correctly. The HPCC Random Access is an *embarrassingly* resilient application and this may be explained by the fact that the computational kernel is simple enough that the HPCC Table T, which is annotated tolerant using our runtime memory allocation programming extension, dominates the active address space mapped in memory during its execution life time.

We are validating the impact of these programming model extensions on scientific applications. The early results from experiments with these scientific application codes, annotated with our proposed programming model extensions suggest that the resilience of these long running scientific HPC applications can be significantly enhanced. This in turn ought to lead to a substantial increase in the checkpointing interval and a reduction in the amount of redundant computation, both of which will reduce the time solution and energy required to solve the most computationally complex and data intensive HPC applications.

References

- [Hukerikar 2012] Hukerikar, S., Diniz, P.C., Lucas, R.F.: A Programming Model for Resilience in Extreme Scale Computing. In: IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W) (June 2012)
- [Kogge 2008] Kogge, P., et al.: ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. In: DARPA (September 2008)
- [Cappello 2009] Cappello, F., et al.: Toward Exascale Resilience International Journal of High Performance Computing Applications (2009)
- [Schroeder 2009] Schroeder, B., Pinheiro, E., Weber, W.-D.: DRAM Errors in the Wild: A Large-scale Field Study. In: Proceedings of the International Conf. on Measurement and Modeling of Computer Systems, SIGMETRICS 2009 (2009)
- [Schroeder 2007] Schroeder, B., Gibson, G.A.: Understanding Failures in Petascale Computers Journal of Physics: Conference Series 78, 012022 (2007)

User Level Failure Mitigation in MPI

Wesley Bland

Innovative Computing Laboratory, University of Tennessee
`wbland@eecs.utk.edu`

1 Introduction

In a constant effort to deliver steady performance improvements, the size of High Performance Computing (HPC) systems, as observed by the Top 500 ranking¹, has grown tremendously over the last decade. This trend, along with the resultant decrease of the Mean Time Between Failure (MTBF), is unlikely to stop; thereby many computing nodes will inevitably fail during application execution [5]. It is alarming that most popular fault tolerant approaches see their efficiency plummet at Exascale [3,4], calling for more efficient approaches evolving around application centric failure mitigation strategies [7].

The prevalence of distributed memory machines promotes the use of a message passing model. An extensive and varied spectrum of domain science applications depend on libraries compliant with the MPI Standard. Although unconventional programming paradigms are emerging, most delegate their data movements to MPI and it is widely acknowledged that MPI is here to stay. However, MPI has to evolve to effectively support the demanding requirements imposed by novel architectures, programming approaches, and dynamic runtime systems. In particular, its support for fault tolerance has always been inadequate [6]. To address the growing interest in fault-aware MPI, a working group in the context of the MPI Forum has proposed basic interfaces and semantics called User-Level Failure Mitigation (ULFM) [1] to enable libraries and applications to survive the increasing number of failures, and, subsequently, to repair the state of the MPIs world. The contributions of this work are at providing a high-level overview of the proposed semantics, and to evaluate the difficulties faced by MPI implementors on delivering a low-impact implementation on the failure-free performance. For a more complete evaluation and discussion of ULFM and its related work, refer to this abstract's accompanying paper [2].

2 Motivation

A major concept followed in the design of any type of parallel programming paradigm, and this in order to provide a meaningful and understandable programming approach, is the avoidance of any potential deadlocks. In addition to this critical requirement the mechanisms involved in the fault management were build with the goal of flexibility, simplicity and performance.

¹ <http://www.top500.org/>

Clearly the most important point, no MPI call (point-to-point or collective) can block indefinitely after a failure, but must either succeed or raise an MPI error. Fault tolerance at the application level is impossible if the application cannot regain full control of the execution after a process failure. The MPI library must guarantee that it will automatically stabilize itself following a process failure, and provide the tools necessary for the application to resolve its own deadlock scenarios on an application specific basis.

Second, the API should allow varied fault tolerant models to be built. MPI has been conceived with the goal of portability and extendability, and have been constructed to support libraries leveraging existing MPI constructs to create more abstractions or tighter integration with libraries. Maintaining this design strength was paramount for ULFM, and it provides this capability so other levels of consistency can be supported as needed by higher-level concepts. Transactional fault tolerance, strongly uniform collective operations, and other FT techniques can all therefore be built upon the proposed set of constructs.

An API should be easy to understand and use in common scenarios, as complex tools have a steep learning curve and a slow adoption by the targeted communities. To this end, the number of newly proposed constructs have been reduced to five (along with nonblocking variants). These five functions provide the minimal set of tools to implement fault tolerant applications and libraries.

Two major pitfalls must be avoided when implementing these concepts: jitter prone, permanent monitoring of the health of peers a process is not actively communicating with, and expensive consensus required for returning consistent errors at all ranks. The operative principle is that fault-related errors are local knowledge, and are not indicative of the return status on remote processes. Errors are raised at a particular rank, when based on local knowledge it is known that a particular operation cannot complete because a participating peer has failed.

3 New MPI Constructs

MPI_COMM_FAILURE_ACK & MPI_COMM_FAILURE_GET_ACKED: These two calls allow the application to determine which processes within a communicator have failed. The acknowledgement function serves to mark a point in time used as a reference for the second function which returns the group of processes which were locally known to have failed. After acknowledging failures, the application can resume MPI_ANY_SOURCE point-to-point operations between non-failed processes, but operations involving failed processes will continue to raise errors.

MPI_COMM_REVOKE: For scalability purposes, failure detection is local to a process's neighbors as defined by the application's communication pattern. This non-global error reporting may result in some processes continuing their normal, failure-free execution path, while others have diverged to the recovery execution path. As an example, if a process, unaware of the failure, posts a reception from another process that has switched to the recovery path, the matching send will never be posted and the receive operation will deadlock. The revoke operation

provides a mechanism for the application to resolve such situations before entering the recovery path. A revoked communicator becomes improper for further communication, and all future or pending communications on this communicator will be interrupted and completed with the new error code `MPI_ERR_REVOKED`.

MPI_COMM_SHRINK: The shrink operation allows the application to create a new communicator by eliminating all failed processes from a revoked communicator. The operation is collective and performs a consensus algorithm to ensure that all participating processes complete the operation with equivalent groups in the new communicator. This function cannot return an error due to process failure. Instead, such errors are absorbed as part of the consensus algorithms and will be excluded from the resulting communicator.

MPI_COMM_AGREE: This operation provides an agreement algorithm which can be used to determine a consistent state between processes when such strong consistency is necessary. The function is collective and forms an agreement over a boolean value, even when failures have happened or the communicator has been revoked. The agreement can be used to resolve a number of consistency issues after a failure, such as uniform completion of an algorithmic phase or collective operation, or as a key building block for strongly consistent failure handling approaches (such as transactions).

4 Implementation Issues

Some of the recovery routines described in Section 3 are unique in their ability to deliver a valid result despite the occurrence of failures. This specification of correct behavior across failures calls for resilient, more complex algorithms. In most cases, these functions are intended to be called sparingly by users, only after actual failures have happened, as a means of recovering a consistent state across all processes. This section describes the algorithms that can be used to deliver this specification and their cost. An evaluation of the failure-free impact on implementations can be found in [2].

Agreement: The agreement can be conceptualized as a failure-resilient reduction on a boolean value. Many agreement algorithms have been proposed in the literature; the log-scaling two-phase consensus algorithm used by the ULFM prototype is one of many possible implementations of `MPI_COMM_AGREE` operation based upon prior work in the field. Specifically, this algorithm is a variation of the multi-level two-phase commit algorithms [9]. A more extensive discussion of the algorithm and its complexity has been published by Hursey, et.al. [8].

Revoke: A concern with the revoke operation is the number of supplementary conditions introduced to the latency critical path. Indeed, most completion operations require a supplementary conditional statement to handle the case where the underlying communication context has been revoked. However, the prediction branching logic of the processor can be hinted to favor the failure free

outcome, resulting in a single load of a cached value and a single, mostly well-predicted, branching instruction, unlikely to affect the instruction pipeline. It is notable that non-blocking operations raise errors related to process failure only during the completion step, and thus do not need to check for revocation before the latency critical section.

Shrink: The Shrink operation is, algorithmically, an agreement on which the consensus is done on the group of failed processes. Hence, the two operations have the same algorithmic complexity. Indeed, in the prototype implementation, MPI_COMM_AGREE and MPI_COMM_SHRINK share the same internal implementation of the agreement.

5 Performance Analysis

A short performance analysis follows which summarizes the efficiency of a representative ULFM implementation based on the development trunk of Open MPI (r26237). A more complete analysis can be found in [2].

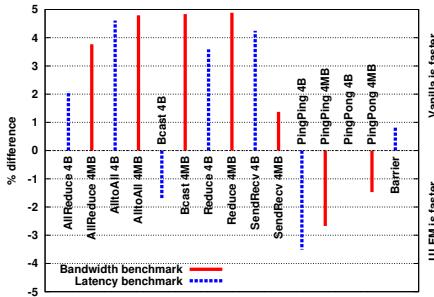


Fig. 1. IMB: ULFM vs. Vanilla Open MPI (Romulus)

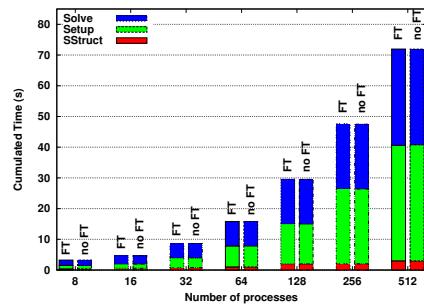


Fig. 2. Sequoia-AMG: ULFM vs. Vanilla Open MPI at various scales (Smoky)

The impact on shared memory systems, sensitive to small modifications of the MPI library, has been assessed on Romulus – a large shared memory machine – using the IMB benchmark suite (v3.2.3) as shown in Figure 1. The duration of all benchmarks remains below 5%, within the standard deviation of the machine’s implementation

To measure the impact of the prototype on a real application, we used the Sequoia AMG benchmark², an Algebraic Mult-Grid (AMG) linear system solver for unstructured mesh physics. A weak scaling study was conducted up to 512 processes following the problem *Set 5*. Figure 2 compares the time slicing of three main phases (Solve, Setup, and SStruct) of the benchmark, with the vanilla implementation of Open MPI, and the ULFM enabled one. The application

² <https://asc.llnl.gov/sequoia/benchmarks/#amg>

itself is not fault tolerant and does not use the features proposed in ULFM. This benchmark demonstrates that a careful implementation of ULFM need not impact the performance of the MPI implementation.

6 Conclusion

Many responsible voices agree that sharp increases in the volatility of future, extreme scale computing platforms are likely to imperil our ability to use them for advanced applications that deliver meaningful scientific results and maximize research productivity. Since MPI is currently, and will likely continue to be – in the medium-term – both the de-facto programming model for distributed applications and the default execution model for large scale platforms running at the bleeding edge, it is the place in the software infrastructure where semantic and run-time support for application faults needs to be provided.

The ULFM proposal is a careful but important step forward toward accomplishing this goal delivering support for a number of new and innovative resilience techniques through simple, familiar API calls, but it is backward compatible with previous versions of the MPI standard, so that non fault-tolerant applications (legacy or otherwise) are supported without any changes to the code. Perhaps most significantly, applications can use ULFM-enabled MPI without experiencing any degradation in their performance, as we demonstrate in this paper. Some of these applications along with other portable libraries are currently being refactored to take advantage of ULFM semantics.

The author would like to acknowledge his co-authors in the full paper [2]: Aurelien Bouteiller, Thomas Herault, Joshua Hursey, George Bosilca, and Jack J. Dongarra.

References

1. Bland, W., Bosilca, G., Bouteiller, A., Herault, T., Dongarra, J.: A proposal for User-Level Failure Mitigation in the MPI-3 standard. Tech. rep., Department of Electrical Engineering and Computer Science, University of Tennessee (2012)
2. Bland, W., Bouteiller, A., Herault, T., Hursey, J., Bosilca, G., Dongarra, J.J.: An Evaluation of User-Level Failure Mitigation Support in MPI. In: Träff, J.L., Benkner, S., Dongarra, J.J. (eds.) EuroMPI 2012. LNCS, vol. 7490, pp. 193–203. Springer, Heidelberg (2012)
3. Bosilca, G., Bouteiller, A., Brunet, É., Cappello, F., Dongarra, J., Guermouche, A., Hérault, T., Robert, Y., Vivien, F., Zaidouni, D.: Unified Model for Assessing Checkpointing Protocols at Extreme-Scale. Tech. Rep. RR-7950, INRIA (2012)
4. Bougeret, M., Casanova, H., Robert, Y., Vivien, F., Zaidouni, D.: Using group replication for resilience on exascale systems. Tech. Rep. 265, LAWNs (2012)
5. Cappello, F., Geist, A., Gropp, B., Kalé, L.V., Kramer, B., Snir, M.: Toward exascale resilience. IJHPCA 23(4), 374–388 (2009)
6. Gropp, W., Lusk, E.: Fault tolerance in Message Passing Interface programs. IJHPCA 18, 363–372 (2004)

7. Huang, K., Abraham, J.: Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers* 100(6), 518–528 (1984)
8. Hursey, J., Naughton, T., Vallee, G., Graham, R.L.: A Log-Scaling Fault Tolerant Agreement Algorithm for a Fault Tolerant MPI. In: Cotronis, Y., Danalis, A., Nikolopoulos, D.S., Dongarra, J. (eds.) EuroMPI 2011. LNCS, vol. 6960, pp. 255–263. Springer, Heidelberg (2011)
9. Mohan, C., Lindsay, B.: Efficient commit protocols for the tree of processes model of distributed transactions. In: SIGOPS OSR, vol. 19, pp. 40–52. ACM (1985)

UCHPC 2012: Fifth Workshop on UnConventional High Performance Computing

Anders Hast¹, Josef Weidendorfer², and Jan-Philipp Weiss³

¹ Uppsala University, Sweden

² Technische Universität München, Germany

³ Karlsruhe Institute of Technology, Germany

Foreword

As the word “UnConventional” in the title suggests, the workshop focuses on hardware and platforms for HPC, which were not intended for HPC in the first place. Reasons could be raw computing power, good performance per watt, or low cost in general. To address this unconventional hardware, often, new programming approaches and paradigms are required to make best use of it. Thus, a second focus of the workshop – for the first time with UCHPC 2012 – is on innovative new programming models for unconventional hardware and how to best combine its computing power with more conventional systems. To this end, UCHPC tries to capture solutions for HPC which are unconventional today but could become conventional and significant tomorrow. While GPGPU still is on topic, especially when integrating the GPUs with multi-cores, other examples for “unconventional” hardware are embedded, low-power processors, upcoming many-core architectures combined with FPGAs or DSPs. Thus, interesting devices for research in unconventional HPC are not only standard server or desktop systems, but also relatively cheap devices due to being mass market products, such as smartphones, netbooks, tablets and small NAS servers. Especially smartphones seem to become more performance hungry every day. Only imagination sets the limits for use of such devices for HPC. The goal of the workshop is to present latest research in how hardware and software (yet) unconventional for HPC is or can be used to reach goals such as best performance per watt. UCHPC also covers corresponding programming models, compiler techniques, and tools.

It is now the 5th time the UCHPC workshop took place, and it is the 3rd time it is co-located with Euro-Par (in 2010, 2011, and 2012). Before that, it was held in 2008 in conjunction with the *International Conference on Computational Science and Its Applications 2008* and in 2009 with the *ACM International Conference on Computing Frontiers 2009*. While the focus of the workshop on the one hand is fixed, it is — on the other hand — a moving target: unconventional for HPC in the beginning was the usage of hardware for game consoles, and especially GPUs. While these still are a hot topic, and get new features for better programmability in every new generation, they rapidly get conventional. This example shows that UCHPC by its main theme needs to regularly re-target

to new research areas with new ideas for HPC. This can be difficult to achieve. While this year, we only have three papers in this post-workshop proceedings volume, they are high quality and we are confident for the future.

From eight submissions, the organizers were able to accept five. Due to a late withdrawal, we finally had four interesting paper talks in the workshop. The reason for only three papers in these proceedings is that one of the papers, while presenting an innovative idea, had to be accepted conditionally on resubmission of an improved paper, which unfortunately did not meet our expectations.

In addition to the paper talks, we were proud to present speakers for two invited talks, making up for a very exciting half-day program. The first session started with the invited talk from Jens Breitbart. It was titled “Programming Models for Next Generation HPC Systems”, representing our additional UCHPC focus on yet unconventional programming models. The main idea of a new PGAS model presented by Jens is to work with memory which only can be written once. This allows to ignore consistency problems with non-coherent shared memory, and hence usage of simpler, power efficient hardware. The session went on with two paper talks. The second session had the same structure, starting with an invited talk by Hartwig Anzt on “Is UCHPC the Solution to the Power Challenge?” He proposed the use of new, power-aware methods not only on the hardware, but also on the software side. As example, he showed that it is possible to relax synchronization requirements in iterative numerical algorithms. Although convergence may be slower, the higher throughput — synchronization on GPUs is expensive — may allow for much faster iterations and thus still faster total run time.

These post-workshop proceedings include the final versions of the presented UCHPC papers which were accepted for publication. They take the feedback from reviewers and workshop audience into account. The organizers want to thank the authors of the papers. Without them, the workshop would not have been able to come up with the interesting topics for discussion. But also, we sincerely thank the EuroPar organization for providing the opportunity to arrange the workshop in conjunction with the Euro-Par 2012 conference, and would like to thank them for a very nice environment. As in the last years, we especially appreciate the hard work of the members of our International Program Committee. They did a perfect job at reviewing the submissions. Last but not least, we thank the — again — large number of attendees this year. They contributed to a lively half-day, and we hope that they found something of interest in the workshop. Based on the positive feedback, the organizers and the steering committee plan to continue the UCHPC workshop in conjunction with EuroPar 2013.

A Methodology for Efficient Use of OpenCL, ESL and FPGAs in Multi-core Architectures

Alexandros Bartzas and George Economakos

National Technical University of Athens
School of Electrical and Computer Engineering
Microprocessors and Digital Systems Laboratory
Heroon Polytechniou 9, GR-15780 Zografou, Athens, Greece
geconom@microlab.ntua.gr

Abstract. OpenCL has been proposed as an open standard for application development in heterogeneous multi-core architectures, utilizing different CPU, DSP and GPU types and configurations. Recently, the technological advances in FPGA devices has turned the parallel processing community towards them. However, FPGA programming requires expertise in a different field as well as the appropriate tools and methodologies. A feasible solution introduced recently is the adoption of ESL and high-level synthesis methodologies, supporting FPGA programming from C/C++. Based on high-level synthesis, this paper presents a methodology to use OpenCL as an FPGA programming environment. Specifically, the opportunities as well as the obstacles imposed to the application developer by the FPGA computing platform and the adoption of C/C++ as input language are presented, and a systematic way to explore both data level and thread level parallelism is given. The resulting methodology can be used for the deployment of parallel applications over a wide range of diverse CPU, DSP, GPU and FPGA multi-core configurations.

1 Introduction

Parallel processing can be considered starting at early 60s, with the D825 from Burroughs Corporation and Dijkstra's paper [3], while notions of parallelism can be found even earlier in the Analytical Engine of Charles Babbage. Since then, a lot of research and development has flourished the field, offering a variety of architectures, programming models, languages and standards. A latest development, mainly since the introduction by the computer gaming industry of ultra-high performance *Graphics Processing Units* (GPUs), is the integration of different processing elements (CPUs, DSPs and GPUs) under a common programming model like CUDA [9] and OpenCL [6].

OpenCL (Open Computing Language) is an open standard for the development of parallel applications on a variety of heterogeneous multi-core architectures, based on the C99 version of the C language. The execution model of OpenCL consists of a *host machine* connected and controlling a *compute device*. The compute device performs calculations with a number of parallel computational intensive *kernels*. Each compute device consists of *compute units* and each

compute unit consists of *processing elements*, where single kernels are executed for specific sets of input data. Communication between processing elements is performed through the memory hierarchy. Each single kernel invocation is also called *work-item*. Work-items are organized into *work-groups*. Work-items have private memory for fast computations while work-groups have a block of local memory, common to all work-items. All work-groups have also access to global memory, which is used for host initiated initialization procedures.

OpenCL, since its introduction, has been reported to support different CPUs, DSPs and GPUs, in a variety of heterogeneous configurations. Recently, the technological advances in *Field Programmable Gate Array* (FPGA) devices, with hundreds of GFLOPs, maximum power efficiency and low cost, has turned the parallel processing community towards them, with a number of publications [11,12,5,8,10] proposing OpenCL as a programming language for FPGAs also. FPGAs, parallel processing and GPUs co-existed for many years, however they were considered isolated and disjoint fields, offering optimizations at different levels in system design. While indeed parallel processing is aiming at a higher optimization level than FPGAs, the main reason for this isolation has been the different programming models and languages used in each case. This is starting to change, as FPGA complexity is requesting higher level programming models for productive exploitation of their capabilities (millions of logic gates, thousands of advanced DSP blocks, hundreds of GFLOPS). Such models start from C level languages and mainly involve *High-Level Synthesis* (HLS) and *Electronic System Level* (ESL) design platforms [2], for the automatic translation of algorithmic into architectural design descriptions.

This paper presents a methodology for the adoption of OpenCL as an FPGA programming environment, based on the ESL platform CatapultC [4]. CatapultC accepts C/C++/SystemC behavioral untimed system descriptions that should follow specific coding guidelines, and through a number of directives (or GUI commands) applies HLS transformations to produce optimized bit-accurate *Register Transfer Level* (RTL) architectural descriptions. The methodology of this paper is a systematic application of each HLS transformation, by a meta-engine placing and tuning CatapultC directives into OpenCL code. The main concern in this process is that even though CatapultC can produce hardware from C, efficient hardware needs effort and architectural synthesis expertise. This expertise is coded in the meta-engine, which iterates through different possible and feasible HLS directive applications to generate optimal hardware implementations of OpenCL kernels. With this approach, the opportunities as well as the obstacles imposed to the application developer by the FPGA computing platform and the adoption of C/C++ as input language are investigated, and a systematic way to explore instruction-level, data-level and thread-level parallelism is given.

2 Related Research

A number of recent publications are considering using parallel programming models (OpenCL and CUDA) as a programming language for FPGAs also.

In [12] the authors present a detailed example, the MyBayes bioinformatics application, of using the CUDA stream-based programming model as an intermediate step for hardware design. All design steps are done manually and no specific methodology is reported. In [8] and [10] two methodologies are given for mapping OpenCL kernels to reconfigurable hardware. The methodologies involve compiler optimizations that map kernel code into fixed hardware templates, which are then written in hardware description languages. While both methodologies are complete and cover many different issues (computations, memory hierarchies and interfacing), the resulting hardware cores are template-based and do not cover in detail lower level design issues. In [5], the authors present another similar methodology, targeting *Application-Specific Processors* (ASPs). They use a custom design environment and map OpenCL kernels into either common or custom ASP instructions. Another approach, closer to this paper is reported in [11], where CUDA code is passed through another HLS tool. Directives and pragmas are used to control the tool but no systematic and iterative application is reported, as in the proposed methodology. HLS is rather considered as a single pass procedure. From the industrial point of view, FPGA vendors have been actively involved in the use of OpenCL for FPGA programming [1], offering specific frameworks that take advantage of the parallelism expressed in OpenCL code and generate template-based FPGA implementations.

3 Translation Methodology

The main idea of this paper is the proposal of a semi-automated methodology to translate OpenCL code into a form suitable for CatapultC, with which hardware is synthesized using HLS. Since OpenCL is based on C99, which is also recognized by CatapultC, this translation does not bring major changes to the input code. The whole process is performed by a custom source-to-source translator, that either infers (if possible) or accepts by the user (this justifies the term semi-automated) details to OpenCL code like pointer sizes, loop boundaries, input parameters and expected return values. The basic steps are the following.

- Each kernel is isolated and HLS synthesizes a hardware components for it.
- Pointers used as formal parameters in functions are converted to arrays with specific dimensions, for correct memory allocation.
- Return values are inserted as formal pointer parameters in the kernel function. This coding technique generates output registers for them.
- Barrier OpenCL instructions are converted into CatapultC I/O transactions with ready/acknowledge interfaces.
- Array sizes are enlarged to reach powers of 2, when feasible. This simplifies synthesis of memory access related hardware.
- Data types are changed into bit accurate and simulation efficient types supported by CatapultC.
- Conditional statements are supplemented so that all mutually exclusive paths are clearly defined. This helps CatapultC schedule them correctly.

- OpenCL specific directives are temporary removed. They are taken into account later, during system integration.
- CatapultC pragmas and directives are inserted. These pragmas and directives control all HLS transformations.

After translation, an iterative procedure is initiated, which works as a meta-engine modifying CatapultC pragmas and directives. At each iteration, which is performed with a predefined scenario (i.e. a loop’s initiation interval is decreased by one in each meta-engine iteration), a new solution is produced. The meta-engine finishes when no new solutions can be produced (further modification of pragmas and directives produces invalid solutions) and the best solution with respect to performance and resource usage is selected for FPGA implementation. In the following subsections details about frequently used pragmas and directives are given.

3.1 Loops

Loops are the main source of optimization in algorithmic synthesis because most computations are performed within loops. Three are the main loop transformations in CatapultC, loop pipelining, loop unrolling and loop merging.

Loop pipelining is controlled by the initiation interval directive. This directive takes a numeric value argument and denotes that each loop iteration will wait that number of control steps before it starts. After that time, the first loop iteration partially overlaps the next and CatapultC generates a pipelined implementation which gets faster as the initiation interval directive gets smaller (minimum value is 1 control step). Our methodology starts with a value equal to the loop iteration latency (larger values would insert idle control steps) and decreases it as long as feasible solutions (valid with respect to the implementation technology) are found. Loop pipelining is a throughput optimization and gives better results if applied in the outer loops in nested loop configurations.

Loop unrolling is controlled by the unrolling directive. Each loop iteration, either unrolled or not, is considered by CatapultC to take at least 1 control step to finish. With unrolling, we investigate the opportunity to put more operations within this limit and lower the repetitions and thus, get faster hardware. Our methodology starts with an unrolling value of 2 and increases it until feasible solutions are found. Very long values tend to serialize the whole loop, which may prevent pipelining, so they are avoided. Loop unrolling is a latency optimization and gives better results if applied in the inner loops of nested loop configurations.

Loop merging can combine loops with identical bounds. Normally, CatapultC schedules consecutive loops one after the other, with no overlapping. If the loops however have identical bounds and data dependencies permit it, both loops can be executed in parallel, by merging their corresponding iterations.

3.2 Memories and Synchronization

CatapultC can map data objects either in register files or in memories. Small data objects can be mapped in register files, with very fast access times but

more complicated control logic. Register files are implemented in FPGAs with *Look-Up Table* (LUT) elements. Large data objects can be mapped in memories with slower access times but less complicated control logic, like dedicated *Block RAM* (BRAM) in FPGAs, or off-chip. To control these options CatapultC uses a threshold directive. Data objects smaller than threshold are mapped into registers while objects larger than threshold are mapped into memories. Moreover, each data object can be forced to be mapped in either type of resource. In our methodology, data objects are mapped both in register files and memories and the best solution is chosen.

Memories in CatapultC have two properties that affect performance, the number of ports (single or dual port memories) and the number of interleaved blocks (1 or more) used. Both properties, controlled by appropriate directives, increase the number of memory accesses in a single control step. On the other hand, complicated memory configurations require complicated control logic. In our methodology both single and dual port memories are selected and interleaving is applied iteratively, until the best solution is found.

OpenCL uses barrier commands for synchronization. In our methodology, each barrier command is converted in a ready/acknowledge signal, which is controlled by the host. A kernel reaching a barrier command sends a ready signal to the host and waits acknowledge. Whenever the host receives ready from all kernels, it responds with the corresponding acknowledge signals.

3.3 System Integration

After all kernels have been synthesized through the proposed meta-engine into hardware blocks, the whole system can be realized (host and kernels). Two modes of execution are supported, as shown in figures 1 and 2. In figure 1, the FPGA device plays the role of a compute device and the host is an external workstation, connected through a high speed interface like the PCI-Express. Multiple FPGAs can be seated in separate boards (one board for each compute device), equipped with a PCI connector. Inside the FPGA, a PCI-Express core is responsible for the communication between the host and the compute device. Alternatively, in figure 2, the whole system (host and compute device) can be realized into the same FPGA, using an embedded processor as host. This solution offers faster connection between host and kernels (i.e a local bus), but since embedded processor ports of OpenCL are not widely available, it can be considered as a future extension. As an exception, in some simple cases, a small controller (manually designed from host OpenCL code) can play the role of the embedded processor.

Summarizing, the proposed methodology can explore different levels of parallelism. During hardware synthesis with CatapultC, instruction-level parallelism is explored by arranging individual instructions for best parallel execution. Data-level parallelism, either as *Single Instruction Multiple Data* (SIMD) or *Single Program Multiple Data* (SPMD) form, is explored by using the same kernel for all work-items and using either shared or local memory. Finally, thread-level parallelism in a *Multiple Instructions Multiple Data* (MIMD) form is explored by synthesizing different kernels and deploying them into different work-groups

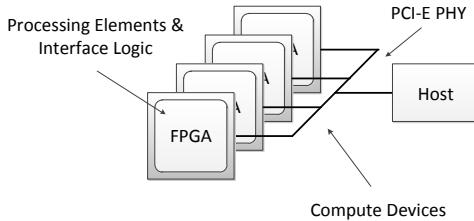


Fig. 1. FPGAs as compute devices

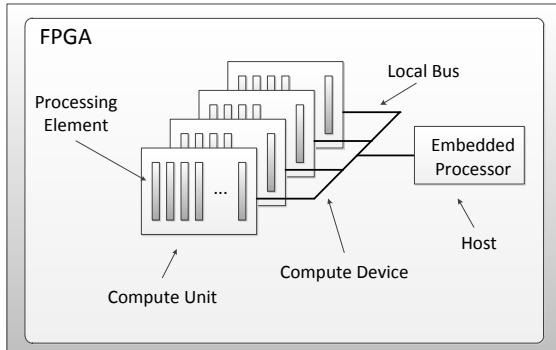


Fig. 2. FPGA as host and compute device

and/or work-items. This wide optimization space leaves many opportunities for system designers to achieve remarkable performance.

4 Experimental Results

The above presented methodology has been tested with a number of OpenCL kernels found in the NVIDIA OpenCL SDK version 4.1. Three kernels have been selected, parallel matrix multiplication, parallel discrete cosine transform (DCT) and parallel inverse discrete cosine transform (IDCT). Parallel matrix multiplication calculates 1 out of an 128x128 result while parallel DCT and parallel IDCT calculate 1 stage out of an 8x8 2 dimensional transformation. Part of the original DCT OpenCL code and the translated CatapultC code are given in the following two listings (omitting `c_x` DCT coefficient and constant declarations), where the great number of similarities as well as the coding guidelines of the previous section can be seen. Specifically, in the CatapultC listing (where all tool specific commands and directives are shown in boldface lines), the bit accurate `ac_int` and `ac_fixed` data types are used, the size of the `D` parameter of the `DCT8` function is inserted and the `hls_design` (meaning that each call corresponds to the same dedicated hardware component and not to an inlined routine), `hls_pipeline_init_interval` and `hls_unroll` directives are inserted with pragma declarations. In the outer `DCT8x8` hardware module, the

`hls_pipeline_init_interval` is equal to 27, which is the minimum value found to generate a feasible solution.

Listing 1.1. OpenCL code

```
inline void DCT8(float *D) {
    float X07P=D[0]+D[7]; float X16P=D[1]+D[6]; float X25P=D[2]+D[5];
    float X34P=D[3]+D[4]; float X07M=D[0]-D[7]; float X61M=D[6]-D[1];
    float X25M=D[2]-D[5]; float X43M=D[4]-D[3];
    float X07P34PP = X07P + X34P; float X07P34PM = X07P - X34P;
    float X16P25PP = X16P + X25P; float X16P25PM = X16P - X25P;
    D[0] = C_norm*(X07P34PP+X16P25PP); D[2] = C_norm*(C_b*X07P34PM+C_e*X16P25PM);
    D[4] = C_norm*(X07P34PP-X16P25PP); D[6] = C_norm*(C_e*X07P34PM-C_b*X16P25PM);
    D[1] = C_norm*(C_a * X07M - C_c * X61M + C_d * X25M - C_f * X43M);
    D[3] = C_norm*(C_c * X07M + C_f * X61M - C_a * X25M + C_d * X43M);
    D[5] = C_norm*(C_d * X07M + C_a * X61M + C_f * X25M - C_c * X43M);
    D[7] = C_norm*(C_f * X07M + C_d * X61M + C_c * X25M + C_a * X43M); }

// 8x8 DCT kernels
__kernel __attribute__((reqd_work_group_size(BLOCK_X,BLOCK_Y/BLOCK_SIZE,1)))
void DCT8x8(__global float *d_Dst,__global float *d_Src,
            uint stride,uint imageH,uint imageW) {
    __local float l_Transpose[BLOCK_Y][BLOCK_X + 1];
    const uint localX=get_local_id(0); const uint localY=BLOCK_SIZE*get_local_id(1);
    const uint modLocalX = localX & (BLOCK_SIZE - 1);
    const uint globalX=get_group_id(0)*BLOCK_X+localX;
    const uint globalY=get_group_id(1)*BLOCK_Y+localY;
    if((globalX-modLocalX+BLOCK_SIZE-1>=imageW) || (globalY+BLOCK_SIZE-1>=imageH))
        return; //Process only full blocks
    __local float *l_V = &l_Transpose[localY + 0][localX + 0];
    __local float *l_H = &l_Transpose[localY + modLocalX][localX - modLocalX];
    d_Src += globalY * stride + globalX; d_Dst += globalY * stride + globalX;
    float D[8];
    for(uint i = 0; i < BLOCK_SIZE; i++) l_V[i * (BLOCK_X + 1)] = d_Src[i * stride];
    for(uint i = 0; i < BLOCK_SIZE; i++) D[i] = l_H[i];
    DCT8(D);
    for(uint i = 0; i < BLOCK_SIZE; i++) l_H[i] = D[i];
    for(uint i = 0; i < BLOCK_SIZE; i++) D[i] = l_V[i * (BLOCK_X + 1)];
    DCT8(D);
    for(uint i = 0; i < BLOCK_SIZE; i++) d_Dst[i * stride] = D[i]; }
```

Listing 1.2. CatapultC code

```
#define INDEX ac_int <32,false>
#define DATA ac_fixed <32,16,true,AC_TRN,AC_WRAP>
#pragma hls_design
#pragma hls_pipeline_init_interval 1
void DCT8(DATA D[BLOCK_SIZE]) {
    DATA X07P=D[0]+D[7]; DATA X16P=D[1]+D[6]; DATA X25P=D[2]+D[5];
    DATA X34P=D[3]+D[4]; DATA X07M=D[0]-D[7]; DATA X61M=D[6]-D[1];
    DATA X25M=D[2]-D[5]; DATA X43M=D[4]-D[3];
    DATA X07P34PP = X07P + X34P; DATA X07P34PM = X07P - X34P;
    DATA X16P25PP = X16P + X25P; DATA X16P25PM = X16P - X25P;
    D[0] = C_norm*(X07P34PP+X16P25PP); D[2] = C_norm*(C_b*X07P34PM+C_e*X16P25PM);
    D[4] = C_norm*(X07P34PP-X16P25PP); D[6] = C_norm*(C_e*X07P34PM-C_b*X16P25PM);
    D[1] = C_norm*(C_a * X07M - C_c * X61M + C_d * X25M - C_f * X43M);
    D[3] = C_norm*(C_c * X07M + C_f * X61M - C_a * X25M + C_d * X43M);
    D[5] = C_norm*(C_d * X07M + C_a * X61M + C_f * X25M - C_c * X43M);
    D[7] = C_norm*(C_f * X07M + C_d * X61M + C_c * X25M + C_a * X43M); }

// 8x8 DCT kernels
#pragma hls_design top
#pragma hls_pipeline_init_interval 27
void ParallelDCT(INDEX stride, INDEX imageH, INDEX imageW, INDEX globalX,
                  INDEX globalY, INDEX localX, INDEX localY, DATA *d_Src, DATA *d_Dst) {
    static DATA l_Transpose[BLOCK_Y][BLOCK_X + 1];
    localY = BLOCK_SIZE * localY; INDEX modLocalX = localX & (BLOCK_SIZE - 1);
    globalX = globalX * BLOCK_X + localX; globalY = globalY * BLOCK_Y + localY;
    if((globalX-modLocalX+BLOCK_SIZE-1>=imageW) || (globalY+BLOCK_SIZE-1>=imageH))
```

```

    return; //Process only full blocks
else {
    DATA *l_V = &l_Transpose[localY + 0][localX + 0];
    DATA *l_H = &l_Transpose[localY + modLocalX][localX - modLocalX];
    d_Src=d_Src+globalY*stride+globalX; d_Dst=d_Dst+globalY*stride+globalX;
    static DATA D[BLOCK_SIZE];
#pragma hls.unroll yes
for(INDEX i = 0; i < BLOCK_SIZE; i++) l_V[i*(BLOCK_X + 1)] = d_Src[i*stride];
#pragma hls.unroll yes
for(INDEX i = 0; i < BLOCK_SIZE; i++) D[i] = l_H[i];
DCT8(D);
#pragma hls.unroll yes
for(INDEX i = 0; i < BLOCK_SIZE; i++) l_H[i] = D[i];
#pragma hls.unroll yes
for(INDEX i = 0; i < BLOCK_SIZE; i++) D[i] = l_V[i * (BLOCK_X + 1)];
DCT8(D);
#pragma hls.unroll yes
for(INDEX i = 0; i < BLOCK_SIZE; i++) d_Dst[i * stride] = D[i]; }

```

Regarding kernel implementation, tables 1, 2 and 3 show the maximum performance achieved for each kernel (as throughput period in ns, the time required before a new input set can be processed by the resulting pipeline architecture) and the required hardware in terms of FPGA resources (Look-Up Table (LUT) function generators, D-type Flip-Flops (DFF), Block RAM (BRAM) and special purpose DSP blocks), both absolute numbers as well as percentages of the maximum available. For all implementations, the largest FPGA of the Xilinx Virtex-6 family was used, the 6VLX760 (with 758784 LUTs, 948480 DFFs, 720x36KB BRAM and 864 DSPs) at 200MHz.

Table 1. Parallel matrix multiplication

Sol.	Perf. (ns)	LUTs	DFFs	BRAMs	DSPs
S1	1295	85 (0.02%)	102 (0.01%)	0 (0.00%)	4 (0.46%)
S2	640	84 (0.02%)	102 (0.01%)	0 (0.00%)	4 (0.46%)
S3	320	113 (0.02%)	118 (0.01%)	0 (0.00%)	8 (0.93%)
S4	160	213 (0.04%)	191 (0.02%)	0 (0.00%)	16 (1.85%)
S5	80	335 (0.07%)	292 (0.03%)	0 (0.00%)	32 (3.70%)

In table 1, the first solution S1 corresponds to no optimizations selected. Solution S2 corresponds to initiation interval set to 1, while solutions S3, S4 and S5 keep this value and add an unrolling factor of 2, 4 and 8 respectively. In tables 2 and 3, solutions S1, S2 and S3 work directly with global memory and utilize fast BRAMs (nonzero in BRAM column), which is a common block for all kernels. This offers advantages at the circuit level (smaller memory controllers, less DFFs) but performance is low because of the large number of global memory accesses (barrier commands blocks every kernel before writing its result). Furthermore, in the same tables, solution S1 corresponds to no optimizations selected, solution S2 corresponds to initiation interval set to 4 (the minimum achieved), solution S3 corresponds to minimum initiation interval and full loop unrolling, solutions S4 and S5 are like S2 and S3 with double width local memories (64 bit I/O ports with 32 bit operands) and solution S6 is like S5 with subfunctions implemented as hardware components and not as inlined code.

In all tables, solution S1 or S2 is the worst implementation. All other solutions are sorted so that each one is better than the previous with respect to performance. Looking at resources, in many solutions less than 1% of the available hardware is used, so there is room to implement large number of kernels. The only resources that limit the number of kernels are the DSP blocks, which increase up to a significant percentage as more parallelization is attempted.

Table 2. Parallel discrete cosine transform

Sol.	Perf. (ns)	LUTs	DFFs	BRAMs	DSPs
S1	455	4158 (0.88%)	1702 (0.18%)	1 (0.14%)	37 (4.28%)
S2	640	4194 (0.88%)	2084 (0.22%)	1 (0.14%)	48 (5.56%)
S3	110	3563 (0.75%)	2354 (0.25%)	1 (0.14%)	23 (2.66%)
S4	30	3649 (0.77%)	2261 (0.24%)	0 (0.00%)	46 (5.32%)
S5	15	5273 (1.11%)	4339 (0.46%)	0 (0.00%)	62 (7.18%)
S6	10	5453 (1.15%)	6292 (0.66%)	0 (0.00%)	64 (7.41%)

Table 3. Parallel inverse discrete cosine transform

Sol.	Perf. (ns)	LUTs	DFFs	BRAMs	DSPs
S1	450	3002 (0.63%)	1688 (0.18%)	1 (0.14%)	38 (4.40%)
S2	800	4703 (0.99%)	2001 (0.21%)	1 (0.14%)	52 (6.02%)
S3	70	3331 (0.70%)	1859 (0.20%)	1 (0.14%)	34 (3.94%)
S4	35	2489 (0.52%)	1519 (0.16%)	0 (0.00%)	54 (6.25%)
S5	15	5329 (1.12%)	4259 (0.45%)	0 (0.00%)	56 (6.48%)
S6	10	5498 (1.16%)	5491 (0.58%)	0 (0.00%)	56 (6.48%)

From the above tables it is shown that a systematic directive application to each kernel code through the proposed meta-engine can produce good quality results in an automated way. In order to get an indication of the overall system performance improvement, we implemented a system based on the architecture of figure 2, with a simple controller to move data from global to local memories, using the slowest solution (S1 in table 2) and the fastest solution (S6 in table 2) of the DCT algorithm. Also, since these solutions differ in resource usage, for S1 16 parallel kernels were mapped onto the FPGA while for S6 8. Image sizes of 256x256, 512x512, 1024x1024 and 2048x2048 were selected and performance (as reported in CatapultC) at a 600MHz clock speed was compared with the

Table 4. Performance comparison between FPGA and GPU

Platform	Execution time (ns)			
	256x256	512x512	1024x1024	2048x2048
Virtex-6(S1)	662102	1216167	2324299	4540563
Virtex-6(S6)	399822	772103	1510840	2988349
Radeon	755398	1225752	2958031	10160484

OpenCL solution found in [7], based on the Radeon HD 6970 GPU at 850MHz. Execution times in ns are shown in table 4.

While the results of table 4 are not 100% objective (not many implementation details are given in [7]), they show a system level speedup ranging from 1.8x (256x256) to 3.4x (2048x2048). Also, S6 (fewer but optimized kernels) is faster than S1 (more but not optimized kernels), which is a justification of our approach. Better results are expected with the Virtex-7 FPGA family (offers twice the resources of Virtex-6), as soon as CatapultC libraries become available.

5 Conclusions

This paper presents a methodology for the adoption of OpenCL as an FPGA programming environment, based on the systematic application of HLS transformations by a meta-engine. The main concern in this process is that even though HLS tools can produce hardware from C, efficient hardware needs effort and some architectural synthesis expertise. This expertise, as shown with experimental results, is captured in the meta-engine, which iterates through different possible and feasible directive applications, and generates optimal hardware implementations. As future extensions, the use of both CUDA and OpenCL under the same environment is considered as well as the use of heuristics in the meta-engine iterations, to speed up the process and produce better results.

References

1. Altera Corporation, <http://www.altera.com/opencl/>
2. Coussy, P., Morawiec, A.: High-level Synthesis: From Algorithm to Digital Circuit. Springer (2008)
3. Dijkstra, E.W.: Solution of a Problem in Concurrent Programming Control. Communications of the ACM 8(9), 569 (1965)
4. Fingeroff, M.: High-level Synthesis Blue Book. Xlibris Corporation (2010)
5. Jaaskelainen, P.O., de La Lama, C.S., Huerta, P., Takala, J.H.: OpenCL-based Design Methodology for Application-Specific Processors. In: 10th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, pp. 223–230. IEEE (2010)
6. Khronos Group, <http://www.khronos.org/opencl/>
7. Kim, C.G., Choi, Y.S.: A High Performance Parallel DCT with OpenCL on Heterogeneous Computing Environment. Multimedia Tools and Applications (2012)
8. Mingjie, L., Lebedev, I., Wawrzynek, J.: OpenRCL: Low-Power High-Performance Computing with Reconfigurable Devices. In: 20th International Conference on Field Programmable Logic and Applications, pp. 458–463. IEEE (2010)
9. NVIDIA Corporation,
http://www.nvidia.com/object/cuda_home_new.html

10. Owaida, M., Bellas, N., Antonopoulos, C.D., Daloukas, K., Antoniadis, C.: Massively Parallel Programming Models Used as Hardware Description Languages: The OpenCL Case. In: International Conference on Computer-Aided Design, pp. 326–333. IEEE/ACM (2011)
11. Papakonstantinou, A., Gururaj, K., Stratton, J.A., Chen, D., Cong, J., Hwu, W.M.W.: FCUDA: Enabling Efficient Compilation of CUDA Kernels onto FPGAs. In: 7th Symposium on Application Specific Processors, pp. 35–42. IEEE (2009)
12. Pratas, F., Sousa, L.: Applying the Stream-Based Computing Model to Design Hardware Accelerators: A Case Study. In: 9th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, pp. 237–246. IEEE (2009)

Efficient Design Space Exploration of GPGPU Architectures

Ali Jooya, Amirali Baniasadi, and Nikitas J. Dimopoulos

Department of Electrical and Computer Engineering
University of Victoria
Victoria, B.C., Canada
{jooya,amirali,nikitas}@ece.uvic.ca

Abstract. The goal of this work is to revisit GPU design and introduce a fast, low-cost and effective approach to optimize resource allocation in future GPUs. We have achieved this goal by using the Plackett-Burman methodology to explore the design space efficiently. We further formulate the design exploration problem as that of a constraint optimization. Our approach produces the optimum configuration in 84% of the cases, and in case that it does not, it produces the second optimal case with a performance penalty of less than 3.5%. Also, our method reduces the number of explorations one needs to perform by as much as 78%.

1 Introduction

Employing Graphics Processing Units (GPUs) as accelerators for general-purpose throughput-intensive workloads has become an important part of high performance computing. GPUs' [1] computing power stems from hundreds of processing elements referred to as PEs. A group of several PEs comprise a Stream Multiprocessor (SM). Multiple SMs are grouped in scalable arrays and connected to memory controllers through an interconnection network. In addition to the PEs, an SM includes three separate L1 caches referred to as constant cache, texture cache and data cache. Other components forming an SM are shared memory, register file and thread pool. Figure 1 shows a typical GPU.

As technology advancements provide designers with a growing chip real-estate, effective methods of exploiting the available resources becomes essential.

The goal of this work is to revisit GPU design and introduce a fast, low-cost and effective approach to optimize resource allocation in future GPUs. Our solution utilizes the Plackett-Burman methodology [2] and formulates a constraint optimization problem to find the optimal GPU design for different available chip real-estate budgets without resorting to exhaustive design space exploration. Our results show that for the applications and configurations investigated, our solution matches the design suggested by an exhaustive search 48 out of 57 times, and for the cases the proposed solution did not match the one found by the exhaustive search, the error is less than 3.5%.

The remaining of this paper is organized as follows. In Section 2 we review the background techniques used. In Section 3 we discuss our methodology. In Section 4 we present the results and we conclude with Section 6.

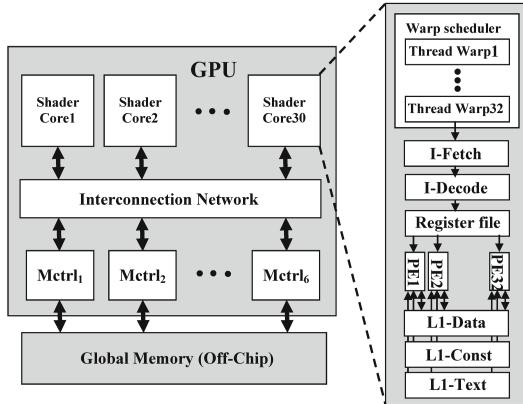


Fig. 1. The structure of a GPU

2 Background

2.1 Plackett-Burman Design Methodology

In design space exploration, one needs to conduct numerous experiments varying a number of parameters to ensure full exploration. In designs with N parameters, each one taking one of L values, L^N experiments need to be conducted. Plackett and Burman [2] introduced a method which can measure the effects of the N parameters, where multi-parameter interaction is not present, by conducting X experiments, where X is the next multiple of 4 strictly greater than N . A Plackett-Burman design with fold-over, can quantify the effect of two interacting parameters. It has been claimed [3] that in the case of processor design exploration with less than 10 parameters, single and two-parameter interactions have the most significant effect, and since we use only four parameters in this work, we shall utilize a Plackett-Burman design with fold-over.

Plackett and Burman [2] introduced templates of experiments for different number of parameters. Table 1 shows an example of such a template (with fold-over). The rows correspond to the experiments while the columns correspond to the parameters. A “+” (“-”) sign indicates that for the experiment indicated, the corresponding parameter is set to its maximum (minimum) value.

For each experiment i , a target performance metric T_i is measured. After running all experiments, it is possible to calculate the effect of each parameter on the design. We denote by S_α the effect of parameter α on the target performance metric. S_α can be calculated by subtracting the target values corresponding to the low parameters from those when they are high. For example, from Table 1, the effect S_E of parameter E is calculated as:

$$S_E = T_1 - T_2 + T_3 + T_4 + T_5 - T_6 - T_7 - T_8 - T_9 + T_{10} - T_{11} - T_{12} - T_{13} + T_{14} + T_{15} + T_{16}$$

The magnitude of the effect is used to identify the parameter that is most important in determining performance.

Table 1. Plackett-Burman design with fold-over; 7 parameters, 16 experiments

experiment	parameters							Target performance	experiment	parameters							Target performance
	A	B	C	D	E	F	G			A	B	C	D	E	F	G	
1	+++ - + - -							T_1	9	- - - + - + +							T_9
2	- ++ + - + -							T_2	10	+ - - - + - +							T_{10}
3	- - + ++ - +							T_3	11	+ + - - - + -							T_{11}
4	+ - - + + + -							T_4	12	- + + - - - +							T_{12}
5	- + - - + + +							T_5	13	+ - + + - - -							T_{13}
6	+ - + - - + +							T_6	14	- + - + + - -							T_{14}
7	++ - + - - +							T_7	15	- - + - + + -							T_{15}
8	- - - - - - -							T_8	16	+ + + + + + +							T_{16}

$S_A \ S_B \ S_C \ S_D \ S_E \ S_F \ S_G$

2.2 Knapsack Problem

The knapsack problem is a constraint optimization problem [4]. Denoting by v_j (w_j) the value (weight) of an item of type j , b_j the maximum number of items of type j and C the capacity of the knapsack, the knapsack problem is stated as:

Select x_j ($j = 1, \dots, n$) items of type j so as to maximize $z = \sum_{i=1}^n v_j x_j$
subject to $\sum_{j=1}^n w_j x_j \leq C$; $1 \leq x_j \leq b_j$ and integer; $j \in N = 1, \dots, n$.

This is a Bounded Knapsack Problem (BKP) [4], a generalization of the 0-1 knapsack problem. We assume that v_j , w_j , b_j and C are positive integers.

3 Methodology

Our objective is to find the GPU configuration that achieves maximum performance under a given transistor budget. Such problem can be solved by an exhaustive design space exploration, which is computationally expensive.

Our approach formulates the problem as a knapsack problem which can be solved efficiently [5]. In formulating the knapsack problem, one needs to identify the *value* (v_j) and *weight* (w_j) parameters. While the *weight* parameter is easily identifiable as the transistor count of each unit, the *value* parameter is not as directly identifiable. The *value* parameter simply shows the contribution of each of the architectural units to the performance of the overall architecture and for a particular application. This information is not readily available. We postulated that within the domain of exploration, the contribution of each unit to the performance is proportional to its cardinality and or size. The proportionality coefficient (i.e. the *value*) though is unknown. In this work, we shall show that the effect S_α of parameter α as per Plackett-Burman, can be used as the *value*.

3.1 Establishing the Transistor Count

Configuring a GPU involves choosing the number of memory controllers, the size of the data cache, the size of the register file and the size of the constant cache among other parameters.

We estimated the cost (in number of transistors) for the data cache, the constant cache and the register file following the method cited in [6], while for the cost of the memory controller, we used the cost of a Xilinx memory controller core [7]. The cost for each unit are shown in Table 4.

3.2 Benchmarks

We used the following set of GPGPU benchmarks from NVIDIA’s CUDA software development kit (SDK) [8] and Rodinia [9] benchmark suits: AES Cryptography (AES); Fast Walsh Transform (FWT); LIBOR Monte Carlo (LIB); 3D Laplace Solver (LPS); Montcarlo; Neural Network Digit Recognition (NN); Scan; Srad; Black schole; Hotspot; Ray tracing (Ray); Matrix multiplication (Matrix); Back propagation (Backprop).

These benchmarks were compiled and run on the GPGPU-Sim [10] simulator with the system configuration shown in Table 2. The ranges of the sizes or cardinality of the units used in configuring the GPU for our experiments, are shown in Table 3. The variation in the values of these parameters resulted in a large number of possible configurations. Totally, we experimented with 553 configurations. The simulations result in performance estimates for each configuration and for each benchmark, reported as *instructions per cycle* (IPC).

3.3 Use of Plackett-Burman and Optimization

In order to determine the *value* parameters, we applied the Plackett-Burman methodology with fold-over for each benchmark. The number of parameters used was four (i.e. number of memory controllers, the size of the data cache, the size of the register file and the size of the constant cache).

The resulting effect values S_α are used as the *value* parameters for the knapsack optimization problem while as the *weight* parameters we used the transistor counts for each of the units as discussed in Section 3.1 above.

We postulated earlier that the contribution of each of the units relates linearly to the performance obtained. Figure 2 shows the performance (as IPC) as a function of the number of memory controllers and for different sizes of the DL1 cache for the LIB benchmark. The linear dependence of the performance on the number of controllers is evident for most of the domain of exploration.

We used MATLAB’s¹ `linprog` function to solve the knapsack problems.

4 Evaluation and Results

As we discussed in Section 3.3 we used four parameters: memory controller block, DL1 cache, constant cache and register file. Table 4 shows the estimated costs of the parameters in terms of transistor counts while Table 3 shows the ranges of the

¹ MATLAB is a registered trademark of The Math Works, Inc. `linprog` is in the optimization toolbox.

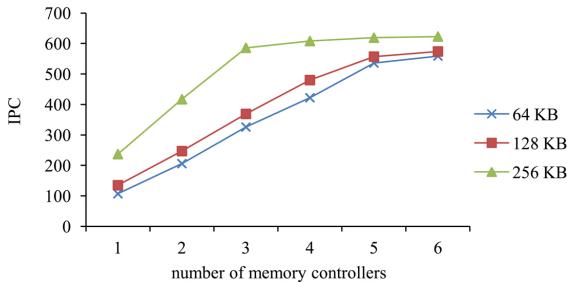


Fig. 2. IPC of LIB benchmark for different sizes of DL1 cache and number of memory controller blocks

Table 2. GPU configuration

Parameter	Value	Parameter	Value
Number of shaders	30	Warp width	32
Shader clock frequency	1.3 GHZ	Scheduling	PDOM
Max thread per shader	1024	max CTA/Shader	8
SIMD pipeline width	32		

configuration parameters. Note that not all benchmarks have constant memory access instructions. Therefore, we have evaluated our method using three or four parameters depending on the benchmark under simulation. We have limited the number of memory controller blocks (MCB) to six, as more than five MCBs do not affect performance for most benchmarks.

We run the optimization problem for a gradually increasing transistor budget and for each budget, we obtain the resulting optimum-performance configuration. Figure 3 shows the resulting configurations for the NN benchmark. The horizontal axis refers to the transistor budget while the vertical axis shows the number of units corresponding to the optimum configuration. In the figure, one can distinguish regions of the transistor budget where the configuration does not change. For the NN benchmark (please refer to Figure 3) we distinguish five regions marked by numbers 1 to 5 on the figure.

As an example, for a transistor budget of no more than 67 million transistors (i.e. within region 2), the optimum configuration comprises four memory controller blocks (MCBs), 2 units of DL1, one unit of Constant Cache and one unit of Register file. Some explanation is necessary here as to the definition of a unit. While the memory controller blocks are self evident, the units of the other parameters vary with the benchmark. For example for the NN benchmark (please refer to Table 3), a unit of DL1 Cache is 8KB, a unit of Constant Cache is 512B while a unit of Register File is 4KB. In this work, we have used as a unit the minimum size considered for each configuration parameter and for each benchmark, and then allowed the optimization function to return as a solution a configuration comprising multiples of the units of each parameter. For the said

example, the optimum configuration obtained comprises 4 Memory Controller Blocks, 16 KB of DL1 Cache, 512B of Constant Cache, and a 4KB register file.

Please note that the caches and register files can only be configured in sizes that are power of two. If the solution of the optimization returned a configuration that had a parameter that violated the “power of two rule”, then the size of this parameter was fixed to a power of two that was less than the size suggested by the optimization, the excess transistor budget was returned to the transistor budget, and another optimization was run on the remaining parameters.

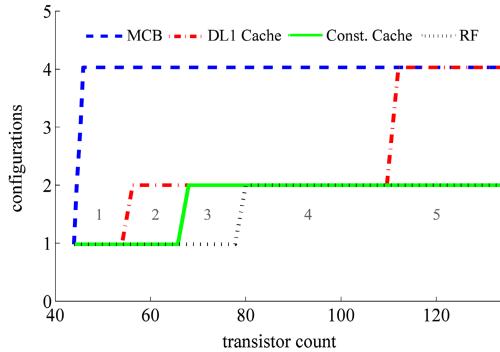


Fig. 3. Optimum configuration regions for the NN benchmark

To ensure that the configurations suggested by the optimization method are indeed optimal, we also performed an exhaustive search of all possible configurations adhering to the transistor budgets considered and compared the results. We shall analyze these results in Section 4.1 below. To conclude the example of the NN benchmark, Table 6 presents all possible configurations that adhere to the transistor budget of 67 million. As it can be verified, the configuration suggested by the optimization solution (Conf12) has indeed the optimum performance. Please note that because of space limitations, this table does not include configurations in Region 1 (i.e. with transistor budgets of less than 56 million). All configurations in region 1 have a lesser performance as compared to Conf12.

4.1 Results

Table 5 shows the Plackett-Burman results for the benchmarks. Each column marked with the name of the parameter reports the effect S_α of the said parameter. The columns marked by “Rank” represent the rank of the effect of the parameter to its left. Thus, for the NN benchmark, we see that the DL1 Cache is ranked as first, that is this benchmark’s performance is influenced primarily by the size of its DL1 Cache. It is interesting to note that for most of the benchmarks, the number of memory controllers influences the performance the most,

Table 3. Benchmarks and low and high values of resources

Benchmark	MCB	DL1 Cache	Const Cache	Register File
AES	1-3	1KB,2KB	512B-8KB	4KB,8KB
FWT	1-4	4KB-32KB	N/A	4KB,8KB
LIB	3-5	32KB-128KB	64KB-256KB	2KB,4KB
LPS	1-3	1KB,2KB	N/A	2KB,4KB
Montcarlo	1-5	32KB,64KB	1KB,2KB	8KB,16KB
NN	1-4	8KB-32KB	512B,1KB	4KB,8KB
Scan	1-3	512B-2KB	N/A	2KB-8KB
Srad	1-6	2KB-256KB	N/A	4KB-16KB
Blackschole	1-4	2K-8K	N/A	4K,8K
Hotspot	1,2	1K,2K	N/A	8K,16K
Ray	1-3	1K,2k	1K-32K	8K,16k
Matrix	1-5	1K,2k	N/A	4K,8K
Backprop	1-3	1K,2k	N/A	4K,8K

Table 4. Transistor cost (in millions of transistors) of resource units

Parameter	Size/Number	Cost	Parameter	Size/Number	Cost
memory controller 1		0.3	Constant cache	32KB	52
DL1 cache	32KB	87	Register file	32KB	170

indicating that the higher memory bandwidth may improve performance significantly, seconded by the size of the DL1 cache, while the sizes of the Constant Cache and the Register File have a lesser impact. However, for some benchmarks, e.g., NN, Ray, LIB and AES, the parameter's rank follows a different order. For example, AES is composed of a single kernel which has 257 blocks of 8 warps. Two blocks can run concurrently on each shader which increases the significance of the number of memory controllers to provide sufficient bandwidth. As AES is optimized to store constants in constant memory the performance is sensitive to the size of constant memory. On the other hand, few local memory accesses reduce the sensitivity of performance to the DL1 cache. LIB runs two kernels, each having 64 blocks of 64 threads. Excessive register file usage of each thread limits the number of concurrent blocks. Similar explanations on the parameter sensitivity can be obtained for the other benchmarks.

Figure 4 shows the performance (as IPC) of the best (grey bars) and ILP-suggested (black bars) configurations for all regions of each benchmark. As it can be seen, the optimization algorithm suggested solutions match the optimum performance ones as found through exhaustive search of all possible configurations, except for nine cases. Namely the solutions suggested by the optimization procedure for Regions 1 and 4 of the AES and Regions 4 and 5 of the Srad and Regions 2, 3, 4, 6 and 8 of the Ray benchmarks, have performance that is slightly lower than the optimal solution found through exhaustive search. For these cases, the performance of the configuration obtained by the optimization

Table 5. Resulting parameter effects after Plackett-Burman for all benchmarks

Bench.	MCB	Rank	DL1 cache	Rank	const cache	Rank	RF	Rank
AES	57100	1	17742	4	35500	2	18922	3
FWT	127799	1	73903	2	—	-	459	3
LIB	1482938	3	2284182	2	1051960	4	6037368	1
LPS	340704	1	136082	2	—	-	16736	3
Montcarlo	445073	1	293537	2	46469	3	26229	4
NN	1054553	2	1508787	1	5307	3	535	4
Scan	12372	1	9708	2	—	-	1748	3
Srad	139159	1	2125	2	—	-	117	3
Blackschole	3110257	1	142613	2	—	-	29369	3
Hotspot	433926	1	133392	2	—	-	4584	3
Ray	34441615	2	38417897	1	1405	4	5684461	3
Matrix	13693	1	7931	2	—	-	1775	3
Backprop	14864	1	12020	2	—	-	476	3

method is very close to the optimum (less than 3.5% difference in performance). Also, in all these cases, the ILP-based solution had the second best performance.

5 Related Work

Wilson et al. [11] proposed a technique to deal with divergent branches by forming new warps dynamically using threads that take different execution paths at the divergence, increasing the system throughput.

Jia et al. [12] proposed an automated GPU performance exploration framework, named Stargazer, based on stepwise regression modelling. Stargazer simulates design points which have been sampled randomly from a full GPU design

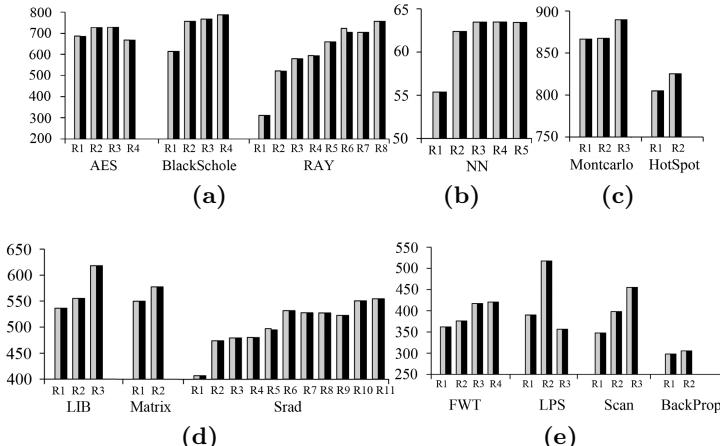


Fig. 4. IPC Comparison between the ILP-suggested and the best performing configurations for different benchmarks

Table 6. All configurations for region 2 of the NN benchmark

parameter	Conf1	Conf2	Conf3	Conf4	Conf5	Conf6
memory controller	1	1	1	1	2	2
constant cache	512B	512B	1KB	1KB	512B	512B
DL1 cache	8KB	16KB	8KB	16KB	8KB	16KB
register file	8KB	4KB	8KB	4KB	8KB	4KB
IPC	39.32	52.71	39.74	52.59	48.78	59.28
cost	65.36	65.86	66.17	66.67	65.66	66.16
parameter	Conf7	Conf8	Conf9	Conf10	Conf11	Conf12
memory controller	2	3	3	3	4	4
constant cache	1KB	512B	512B	1KB	512B	512B
DL1 cache	8KB	8KB	16KB	8KB	8KB	16KB
register file	8KB	8KB	4KB	8KB	8KB	4KB
IPC	48.90	52.83	61.55	52.49	54.98	62.38
cost	66.47	65.96	66.46	66.77	66.26	66.76

space and uses these sample simulation to build a performance estimator. The results showed that using 0.03% of full design space, Stargazer predicts the performance of any design point with less than 1.1% error.

Palermo et al. [13] proposed a DSE methodology for application-specific multi-processor system-on-chip. The proposed methodology uses design of experiment techniques, e.g. random and full factorial, to find a set of good candidate architecture configurations to minimize the number of simulations. Then using response surface modeling and the simulation results, an analytical representation of the system's objective function is generated.

Couvreur et al. [14] applied MMKP to design a MP-SoC runtime manager. Whenever the environment is changing, the runtime manager considers the specification of available application mappings, information provided by the design time exploration, the platform information, and the user requirements to select operating points that minimize total energy consumption.

Lilja et al. [3] applied Plackett and Burman design to identify key processor parameters and analyzed their effect on processor enhancement.

6 Conclusions

In this work, we presented a method that establishes a GPU configuration that adheres to a transistor budget limit and obtains the optimum or near optimum performance. Among all parameters in GPU design space, we used four parameters, i.e. register file size, shared memory and data cache sizes and number of memory controller, that we could estimate the costs in term of transistor count. We did not include the remaining of the parameters, because of the difficulty in calculating the costs.

Our method delivered the optimum performing configuration in 48 out of 57 cases, and for the nine cases where the configuration did not achieve optimum

performance, its performance lagged the optimum one by less than 3.5%. Our method is very efficient in that it requires far fewer simulations to achieve its results. For example, for the case of the Srad benchmark, an exhaustive design exploration would have required the simulation of 144 configurations, while our method arrives at the optimum configuration by simulating just 16.

Further, we plan to investigate the applicability of the method discussed in this work to designs where the constraint is power rather than transistor count.

Acknowledgment. Computational support was provided through Compute/Calcul Canada. This work was supported in part by NSERC, the Lansdowne Chair in Computer Engineering at UVic, and IBM through a Faculty Award.

References

1. Nickolls, J., Dally, W.J.: The GPU Computing Era. *IEEE Micro* 30(2), 56–69 (2010)
2. Plackett, R., Burman, J.: Design of optimum multifactorial experiments. *Biometrika* 33(4), 305–325 (1944)
3. Yi, J.J., Lilja, D.J., Hawkins, D.M.: Improving computer architecture simulation methodology by adding statistical rigor. *IEEE Transactions on Computers* 54(11), 1360–1373 (2005)
4. Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons (1990)
5. Parra-Hernandez, R., Dimopoulos, N.J.: A new heuristic for solving the multichoice multidimensional knapsack problem (MMKP). *IEEE Trans. on Systems Man and Cybernetics Part A* 35(5), 708–717 (2005)
6. Steinhause, M., Kolla, R., Larriba, J., Ungerer, T., Valero, M.: Transistor count and chip-space estimation of simplescalar based microprocessor models. In: Workshop on Complexity-Effective Design (2001)
7. Spartan: LogiCORE IP multi-port memory controller (v6.04.a) (July 6, 2011)
8. NVIDIA: CUDA SDK code samples (v4)
9. Che, S., Boyer, M., Meng, J., Tarjan, D., Lee, S., Sheaffer, J.W., Skadron, K.: A benchmark suite for heterogeneous computing. In: IEEE International Symposium on Workload Characterization (October 2009)
10. Bakhoda, A., Yuan, G., Fung, W.W.L., Wong, H., Aamodt, T.M.: Analyzing CUDA workloads using a detailed GPU simulator. In: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) (April 2009)
11. Fung, W., Sham, I., Yuan, G., Aamodt, T.: Dynamic warp formation and scheduling for efficient GPU control flow. In: 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2007, pp. 407–420 (December 2007)
12. Jia, W., Shaw, K.A., Martonosi, M.: Stargazer: Automated regression-based GPU design space exploration. In: IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2012 (2012)
13. Palermo, G., Silvano, C., Zaccaria, V.: Respir: A response surface-based pareto iterative refinement for application-specific design space exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28(12), 1816–1829 (2009)
14. Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In: International Symposium on System-on-Chip 2006, pp. 1–4 (November 2006)

Spin Glass Simulations on the Janus Architecture: A Desperate Quest for Strong Scaling

M. Baity-Jesi^{1,2}, R.A. Baños^{2,3}, A. Cruz^{2,3}, L.A. Fernandez^{1,2},
J.M. Gil-Narvion², A. Gordillo-Guerrero⁷, M. Guidetti², D. Iñiguez^{2,8},
A. Maiorano^{9,2}, F. Mantovani⁵, E. Marinari^{9,10}, V. Martin-Mayor^{1,2},
J. Monforte-Garcia^{2,3}, A. Muñoz-Sudupe¹, D. Navarro³, G. Parisi^{9,10},
S. Perez-Gavirio², M. Pivanti⁹, F. Ricci-Tersenghi^{9,10}, J. Ruiz-Lorenzo^{7,2},
S.F. Schifano⁶, B. Seoane^{1,2}, A. Tarancón^{2,3}, P. Tellez³, R. Tripiccione⁶,
and D. Yllanes^{9,2}

¹ Universidad Complutense, Madrid, Spain

² Instituto BIFI, Zaragoza, Spain

³ Universidad de Zaragoza, Zaragoza, Spain

⁴ Università di Tor Vergata and INFN, Roma, Italy

⁵ Universität Regensburg, Regensburg, Germany

⁶ Università di Ferrara and INFN, Ferrara, Italy

⁷ Universidad de Extremadura, Badajoz/Cáceres, Spain

⁸ Fundacion ARAID, Zaragoza, Spain

⁹ Università La Sapienza, Roma, Italy

¹⁰ IPCF-CNR and INFN

Abstract. We describe Janus, an application-driven architecture for Monte Carlo simulations of spin glasses. Janus is a massively parallel architecture, based on reconfigurable FPGA nodes; it offers two orders of magnitude better performance than commodity systems for spin glass applications. The first generation Janus machine has been operational since early 2008; we are currently developing a new generation, that will be on line in early 2013. In this paper we present the Janus architecture, describe both implementations and compare their performances with those of commodity systems.

1 Introduction

A major challenge in condensed-matter physics is the understanding of glassy behavior. Glasses are materials with strong industrial relevance (aviation, pharmaceuticals, automotive, etc.) that do not reach thermal equilibrium in human lifetimes; a quantitative understanding of their behavior is still an open problem.

Spin glasses (SG) [1] are a widely studied category of prototypical glassy models. Spin variables, taking a small set of discrete values (e.g. just two, ± 1) sit at the nodes of a regular D -dimensional lattice, and tend to align (or anti-align) with their neighbors according to the sign of a coupling constant associated to the link connecting neighbor sites.

The deceptively simple rules (see later) governing the evolution of each spin, translate into complex collective dynamics as soon as the lattice has even a fairly small number of sites (e.g., ≥ 1000 sites). A striking feature of glasses – and of SG models – is that at low enough temperatures (below the *critical temperature*, T_c) important material properties, such as the compliance modulus or specific heat, depend on time even if the sample is kept for months (years) at constant experimental conditions.

If one quickly cools a sample below T_c , glassy domains grow in the system; their size defines a time-dependent coherence length $\xi(t)$; $\xi(t)$ remains very small (e.g., just a few tens of lattice spacings) even for macroscopic times. Good news is that lattices whose size is just a small multiple of $\xi(t)$ reproduce the experimental evolution, so numerical simulations on computationally affordable lattices are an accurate investigation tool. Bad news is that long simulations are necessary. One usually studies SG with Monte Carlo techniques; one Monte Carlo step (the trial of one spin-reversal on all nodes of the lattice), roughly relates to the average spin-flip time in a real sample, $\simeq 1$ ps; real experiments span a time scale of seconds, that is $\geq 10^{12}$ complete lattice updates in the simulated dynamics. If one wants to simulate $\simeq 100$ independent samples of a lattice with 100^3 sites for 1 (equivalent) second the computational burden quickly goes to 10^{20} spin-flip trials. One may also want to study spin glass properties at equilibrium, for which $\xi(t)$ is roughly the size of the system; this is only possible today for small lattices (e.g. of linear size ≤ 32); thermalization may require in this case 10^{12} Monte Carlo steps on thousands of independent samples.

We see that SG simulation is a computational grand challenge. Luckily enough, SG are easy to treat numerically as their computational algorithms (see Section 2) offer a huge amount of easily identified parallelism; however, the size of our simulated system is (and must remain) of limited size: as more computing resources become available, they must speed up the simulation of a problem of (almost) fixed size; we are in a regime governed by Amdahl's law, as opposed to exploiting larger resources to tackle larger problems (where Gustafson's law applies). If one uses standard parallel systems this means that *strong* performance scaling is crucial: this is why an application-driven system is appropriate.

This paper describes *Janus*, a reconfigurable architecture carefully optimized for Monte Carlo simulations of SG. The first generation Janus machine, deployed in early 2008, outperformed standard computing systems by two orders of magnitude; after four years it still retains a non-negligible performance edge. We are now developing a new generation – *Janus2* – planned for early 2013; again we expect performances $\mathcal{O}(100)$ better than commercial systems.

This paper is organized as follows: we first describe the SG models we are interested in, and the Monte Carlo techniques used to investigate them; we then describe the Janus architecture, highlighting the features that make it a very efficient SG number-cruncher. We then discuss the first Janus version and provide details on the new implementation, Janus2. Next, we compare (measured) Janus and (expected) Janus2 performance with other options available over the lifetime of the project. Our conclusions and outlooks end our text.

2 Spin Glass Models and Their Simulation

We start by introducing a widely studied SG model and its associated computational algorithms, highlighting their architectural implications.

The three-dimensional Edwards-Anderson model [2] is a popular SG model, defined on a lattice of L^3 sites; it is easily described in terms of the energy of the system:

$$E = - \sum_{\langle ij \rangle} \sigma_i J_{ij} \sigma_j; \quad (1)$$

σ_i are L^3 spin variables (modeling the magnetic moments at atomic sites); they take values ± 1 and sit at the nodes of the lattice; the sum spans all pairs of nearest neighbors in the lattice. J_{ij} are the strengths of the interaction (couplings) along the edges connecting nearest-neighbor sites; $J_{ij} > 0$ favors alignment of the corresponding spins, a negative value favors misalignment. J_{ij} values are usually extracted from a distribution with zero mean and unit variance; the simplest case is that $J_{ij} = \pm 1$ with equal probability (binary model). A given assignment of $3L^3 J_{ij}$ defines a *sample* of the system;

The *local energy* of a spin at site k is $\epsilon(\sigma_k) = -\sigma_k \phi_k$; ϕ_k depends on the value of the neighbor spins:

$$\phi_k = \sum_{j=k \pm x, k \pm y, k \pm z} J_{kj} \sigma_j; \quad (2)$$

for a given configuration of neighbors, $\epsilon(\sigma_k)$ is two valued, $\epsilon(\pm 1) = \mp \phi_k$. With the assumption that σ_k is at equilibrium with its neighbors at a given temperature T , the probabilities that $\sigma_k = \pm 1$ are given by the Boltzmann-Gibbs distribution,

$$P(\sigma_k = 1) = \frac{\exp [\beta \phi_k]}{\exp [\beta \phi_k] + \exp [-\beta \phi_k]}. \quad (3)$$

$\beta = 1/T$ is the *inverse temperature* (in units such that the Boltzmann constant K_B equals 1). This defines the *Heat-Bath* Monte Carlo algorithm (see ref. [3] for a review on Monte Carlo algorithms): we may decide if the spin σ_k is up or down by comparing the probability, Eq. (3), with a (pseudo-)random number ρ , uniformly extracted in $[0, 1)$.

The simulation of a sample starts from an arbitrary initial configuration (usually chosen by randomly assigning ± 1 to all spins) and proceeds as follows:

1. begin a trial spin-flip: pick a site k at random, with uniform probability;
2. compute ϕ_k , Eq. (2), and the spin up probability $P(+1)$, Eq. (3);
3. pick a uniformly distributed pseudo-random number $0 \leq \rho < 1$;
4. if $\rho < P(+1)$, then $\sigma_k = +1$, otherwise $\sigma_k = -1$; end of the trial spin-flip;
5. repeat as many times as needed;

One easily maps spin values to bits by setting $\sigma_k \rightarrow S_k = (1 + \sigma_k)/2$ and applying similar transformations for $J_{ij} \rightarrow \hat{J}_{ij}$ and $\phi_k \rightarrow F_k$; most of the processing then reduces to logical (as opposed to arithmetic) operations on bits. F_k takes only

seven integer values in $[0 \dots 6]$ so the values of $P(+1)$ can be precomputed and stored in a small look-up table, addressed by F_k .

A Monte Carlo step (MCS) is defined as a number of trial spin-flips equal to the number of sites in the lattice; one such step relates to the average spin-flip time for real systems, as discussed in the introduction. Each MCS produces a new spin configuration on the lattice; one can show that the sampled configurations asymptotically follow the Boltzmann-Gibbs distribution (although, in practice the number of required MCS may be far larger than possible). One also shows (see e.g., [3]) that, for a large number of MCS, the statistical properties of the observables (averages over all sites and over MCS) do not depend on the order in which the algorithm visits each lattice sites; one then adopts a fixed sequence, so each site is visited once and only once in each MCS, and always in the same order. This makes it easy to efficiently exploit available parallelism, as we will see shortly.

The algorithms described above are extremely compute-friendly:

- there are only two critical kernels in the computation, the assignment of a new value to σ_k and the correlated generation of one random number;
- both kernels are based on a small number of logical and arithmetic operations on a small set of variables;
- the main computational structure, when repeated over the whole lattice, gives rise to regular loops performing the same set of operation on a regularly structured data base, whose elements are stored in memory in regular patterns that do not depend on the computation itself;
- Control is simple and memory addressing is regular: simple state-machines are enough for both purposes;
- there is a huge amount of available internal parallelism; this is most easily unveiled, by considering a checkerboard decomposition of the lattice: the neighbors of all black sites are white, so each of the two subsets can in principle be operated upon at the same time;
- there is an additional level of available parallelism – external parallelism
 - associated to the fact that one wants to accumulate statistics on several unrelated samples, or to perform simulations at many different temperatures.

The last point is easily exploited by farming; previous points define the challenge for an efficient SG engine: extract the largest possible amount of available parallelism; The ideal SG machine can be seen as a large collection of identical cores, that perform efficiently the small set of needed logical and arithmetic operations; a single control structure drives all cores; they work concurrently, performing the same thread at the same time. Each core is a slim object, of just $\simeq 1000$ logical gates, so thousands of them can be easily deployed.

Another way to look at the ideal SG machine is to view it as an *application specific* GPU, with data paths tailored to the specific sequence of logical operations, a control structure shared by a number of cores larger than in state-of-the-art GPUs, variables allocated on on-chip memory and a memory controller optimized for the access patterns required by the chosen algorithm.

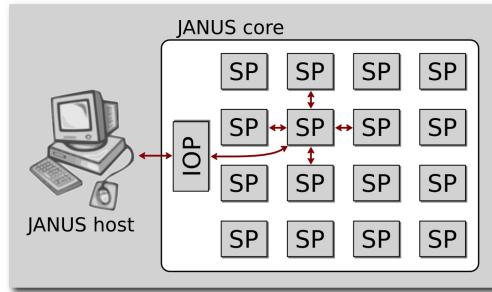


Fig. 1. Conceptual architecture of a Janus system, including the cluster of Simulation Processors (SPs), the Input-Output Processor (IOP) and the host PC.

3 Janus Architecture

As pointed out above, an efficient SG machine is a large collection of *spin-processors*. The spin-processor is a small computing element, tailored to the required mix of operations on bit-valued variables, and to random number generation. Currently available commodity processors are at large variance with these requirements, mainly because their architectures are optimized for long data words. This makes the logic complexity of each data path much larger than needed in SG simulations, and also makes it hard to map efficiently spins onto machine words.

Over the years, *Field Programmable Gate Arrays* (FPGA) have steadily become more powerful and flexible enough to become key building blocks for an SG-computer. FPGAs are reconfigurable at gate level, and the amount of available resources in state-of-the-art FPGAs is enough to accommodate a large cluster of spin-processors. Last but not least, storing the lattice on on-chip memory makes it possible to provide the large memory bandwidth needed to keep all spin-processors at work.

The available parallelism of SG applications is larger than can be exploited by one FPGA: a set of FPGAs can be connected together by a simple network, e.g. a first neighbor torus, and one can process SG samples on this multi-node structure.

A host computer is connected to this structure: contrary to several recent proposed reconfigurable machines [4–6], the host has a minor role, and it is only needed to initialize and start the FPGA array, and to collect simulation results; relatively low bandwidth and long latency are acceptable in our case.

The architecture of the Janus machine [8–10] follows these guidelines; figure 1 shows the Janus concept. Janus is a cluster of computing elements called SP and based on FPGAs; the smallest Janus block is based on 16 FPGAs mounted on a processing-board; our largest installation has 16 processing-boards mounted in a standard 19" rack and controlled by 8 host PCs. Each processing-board hosts a communication hub (the IOP, also using an FPGA) between the SPs and the host PC; communication is based on a gigabit Ethernet interface.



Fig. 2. Left: a Janus processing-board with 16 SPs; Right: a large Janus system with 16 processing-boards, 256 SPs and 8 host PCs.

The IOP handles input-output operations between the SPs and the host PC, and acts as a supervisor for all SPs.

Janus uses Xilinx Virtex-4 LX200 FPGAs, the largest available elements of a state-of-the-art FPGA family when the system was built. The main advantage of FPGAs is that they allow a quick and low cost development cycle; a further step would be to develop an hardwired application-specific processor, as done, for instance, in the Anton project [7]. The main clock is fixed once for all at a conservative 62.5 MHz. This choice reduces the effort needed to successfully map on the FPGA many different and quickly evolving versions of our codes, associated to various possible choices for the physics model and the simulation algorithm.

Figure 2 is a picture of the system. Janus was developed in 2006-2007; it helped obtain significant physics results, as early as mid 2008 [11]; a recent summary is presented in [12].

User applications running on Janus are made of two parts: a firmware module, that defines the operation of the SPs, and a C program running on the host PC; the C program performs data initialization, data input/output and supervises the operations of the SPs. Firmware modules are written in VHDL and compiled by the appropriate Xilinx tools; FPGAs on the SPs are configured on a run-by-run basis, at the start of each application run. Using VHDL, our heaviest SG applications implement up to 1024 spin-processors on each FPGA (and a matching number of random number generators; we use throughout the simple but very efficient Parisi-Rapuano generator [13]); they use approximately 95% of the resources of the FPGA; the actual data-path is very complex and critically exploits the large number of on-chip memory blocks, providing the needed memory bandwidth; for full details, see [9, 12].

A more user-friendly programming environment is not a goal of the Janus project; some limited experimentation with high level FPGA programming tools ended with only $\leq 10\%$ of the performance of VHDL codes. This scenario – acceptable for JANUS, as only a small set of applications, each running for weeks or months, is needed – is a key problem for more general purpose FPGA machines, like Novo-G [6], whose architecture is not too different from Janus.

Looking again at (strong) performance scaling, each Janus SP is an ideal device, as performance scales linearly as more cores are added within each FPGA. This trend ends as we try to map one physical lattice over several SPs, as the network bandwidth ($\simeq 1\text{Gbit/s}$ per link) is not enough (by a factor 3 if we map an 80^3 lattice on a full Janus board).

4 Janus2

We have recently started the development of a new Janus generation, that we call *Janus2*. Its architecture follows the approach described above, with several changes made possible by recent advances in FPGA technology. We plan the following architectural and technology improvements:

- we make the torus network 3D, so one lattice sample can be mapped onto a larger number of SPs; this allows faster processing of a given lattice size, but also simulations of much larger systems;
- we add fast DDR3 memory to each SP node; this is not necessary for SG simulations; however with this improvement Janus2 becomes a more flexible re-configurable system; for instance, we are already considering graph-coloring algorithms for which the old Janus was poorly suited as enough memory was not available;
- we place the host CPU closer to the SP array, using a PCI-Express (PCIe) interface; this increases bandwidth by a factor 40 and decreases latency to $\approx 1\mu\text{s}$, allowing a much closer control of the SP array by the host;
- the IOP module is directly connected to the host PC through an 8X PCI-Express bus; the host PC is a Computer-On-Module (COM) system, directly plugged onto the processing-board;
- we use the VX485T device of the latest Xilinx Virtex 7 FPGA family. This more than doubles the complexity that can be mapped, and preliminary synthesis have verified that the clock frequency of our codes can be increased by a factor 4X; all in all we expect that each Janus2 SP will be 8 times faster than Janus.

As before 16 SPs will be installed on a mother-board, arranged on the edges of a $4 \times 4 \times 1$ 3D-grid. All links of the torus network use high speed serial links directly available on the FPGA. We plan ≥ 20 Gbit/sec for the on-board links and ≥ 5 Gbit/sec for the links in the Z-directions (that run on cables across multiple processing-boards). These figures match the bandwidth requirement of linear scaling for a lattice of 160^3 points, that we plan to parallelize on all 16 Janus2 SPs on a board; so, we expect an overall *strong scaling* performance

Table 1. Spin-update-time (SUT) of EA simulation codes on a 64^3 lattice on several architectures. CBE is a system based on the IBM Cell processor; Tesla C1060 is an NVIDIA GP-GPU with 448 cores; NH (SB) are dual-socket systems based respectively on the 4-core Nehalem Xeon-5560 (8-core Sandybridge Xeon-E5-2680) processors.

L^3	Core 2 Duo	CBE (16 cores)	Janus	Tesla C1060	NH (8 cores)	SB (16 cores)	Janus 2
2007	2007	2008	2009	2009	2012	2013	
64^3	1000 ps	150 ps	16 ps	720 ps	200 ps	60 ps	2 ps

increase of a factor ≥ 100 . We will also consider much larger lattices, for which a real 3D system with several processing-boards will be used. We expect working prototypes of this system in late summer, this year.

5 Janus Performances

In this section we analyze Janus performances, comparing them with conventional systems based on recently developed multi- and many-core CPU architectures.

We first consider conventional performance figures: a reference Monte Carlo algorithm for the simulation of the EA model performs the following operations for each lattice site:

- generation of 1 random number; using the Parisi-Rapuano generator this step performs 1 32-bit sum and 1 32-bit XOR;
- computation of the local field; 6 3-bit XORs and 5 3-bit sums;
- test of the Heat-Bath probability: one 32-bit comparison.

Conservatively equating the 6 short xor and sum operations to one 32-bit integer operation, we end up with 6 equivalent operations for spin update. On Janus each SP has 1000 62.5 MHz spin-processors; each SP then delivers $\simeq 375$ Giga-ops (that is $\simeq 96$ Tera-ops for our large system); Janus has also been a very energy efficient machine, at $\simeq 10$ Giga-ops/W (for comparison, the top entry of the Green500 list in summer 2008 had $\simeq 500$ Mflops/W).

We now consider a performance metrics relevant to the physics user; the *System spin Update Time* (SUT) is the average time needed to update *one* spin of *one* lattice. For each SP in Janus SUT is 16 ps, and we estimate that it will decrease to 2 ps for Janus2; for one full Janus2 processing board working on one lattice, SUT goes down to 0.125 ps.

When comparing with standard computers, one also introduces a *Global spin Update Time* (GUT), appropriate when one simulation job handles several samples of the lattice at the same time; GUT is the SUT value divided by the number of lattices simulate in parallel. This slightly awkward definition is appropriate, since one has been forced, when using traditional CPUs, to combine the variables of several independent samples into the bits of one processor word and update those samples in parallel, in order to boost overall performance; this

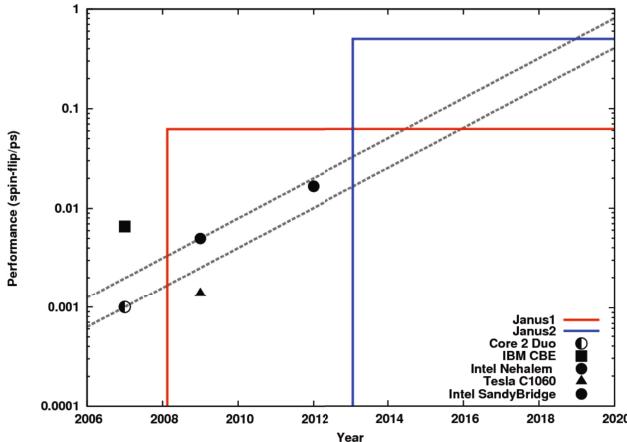


Fig. 3. EA performance (spin-flip/picosec) for optimized programs on several architectures as a function of time. The dotted grey lines scale according to Moore’s law. See the text for more details.

trick, usually called multi-spin coding improves the amount of statistically relevant information made available by a run in a given wall-clock time, but does not improve on the time needed to perform a given number of MCS.

When the Janus project started, early 2006, state-of-the-art commodity systems had dual-core CPUs; on those processors carefully optimized codes had a SUT of $\simeq 1000$ ps and GUT of $\simeq 400$ ps. In the following years, processors have changed significantly with the introduction of many-core CPUs and of general purpose GPUs; these are better SG machines than traditional CPUs as one maps the available parallelism on more cores (or on more threads, for GPUs).

Over the years, we have compared [14, 15] Janus with several multi-core systems. A summary of results is collected in Table 1. We clearly see that over the years the large performance gap of Janus has been significantly eroded; an interesting first example was the extremely efficient IBM Cell CPU, for which SUT is 150 ps. In April 2012 the best figure is offered by a 16 cores Sandybridge-based system, for which SUT is ≈ 60 ps.

6 Conclusions and Outlook

We have described two generation Janus machines, that we have developed for Spin Glass applications, and analyzed their measured and expected performances.

An obvious question when developing a custom system is how long it will keep its performance edge over commercial systems. An educated guess to this question for Janus2 comes from Figure 3 that graphically presents part of the data of Table 1. A reasonably clean pattern emerges:

- while commercial machines increase in performance over the time in a regular way, application-specific projects imply a sequence of step functions over time, as there is no performance gain till a new generation is available.
- Intel processor performance has grown faster than expected by Moore's law.
- we interpret this fact as the consequence of a performance gap that happened when multi-core processors were introduced, followed by a regular Moore's behavior (compare the two Moore's lines in the picture).
- pending new architectural changes, Janus2 should remain a competitive simulation engine at least up to the year 2017.
- a by-product of our analysis shows the poor performance of GPUs for this problem, as well as the outstanding performance of the IBM-Cell processor, whose production has however been discontinued.

Acknowledgments. Janus was supported by the EU (FEDER funds, UNZA05-33-003, MEC-DGA, Spain), by MEC (Spain) (FIS2006-08533, FIS2007-60977, FIS2010-16587, FIS2009-12648-C03, FPA2004-02602, TEC2007-64188, TEC2010-19207), by CAM (Spain), by Junta de Extremadura (GR10158) and by the Microsoft Prize 2007. M. Pivanti was supported by ERC grant agreement N.247328.

References

1. Mézard, M., Parisi, G., Virasoro, M.A.: Spin Glass Theory and Beyond. World Scientific, Singapore (1987); Young, A.P. (ed.): Spin Glasses and Random Fields. World Scientific, Singapore (1998)
2. Edwards, S.F., Anderson, P.W.: J. Phys. F: Metal Phys. 5, 965–974 (1975); ibid. 6, 1927–1937 (1976)
3. Sokal, A.D.: Functional Integration: Basics and Applications (1996 Cargèse School). In: DeWitt-Morette, C., Cartier, P., Folacci, A. (eds.). Plenum, New York (1997)
4. Baxter, R., et al.: Maxwell - a 64 FPGA Supercomputer. In: Second NASA/ESA Conference on Adaptive Hardware and Systems, pp. 287–294 (2007)
5. Flynn, M., et al.: Finding Speedup in Parallel Processors. In: International Symposium on Parallel and Distributed Computing, ISPD 2008, pp. 3–7 (2008)
6. George, A., Lam, H., Stitt, G.: Computing in Science & Engineering 13, 82–86 (2011)
7. Shaw, D.E., et al.: Comm. ACM 51, 91–97 (2008)
8. Belletti, F., et al.: Computing in Science & Engineering 8, 41–49 (2006)
9. Belletti, F., et al.: Comp. Phys. Comm. 178, 208–216 (2008)
10. Belletti, F., et al.: Computing in Science & Engineering 11, 48–58 (2009)
11. Belletti, F., et al.: Phs. Rev. Lett. 101, 157201 (2008)
12. Baity-Jesi, M., et al.: Eur. Phys. J. Special Topics 210, 33–51 (2012)
13. Parisi, V., Parisi, G., Rapuano, F.: Phys. Lett. B 157, 301 (1985)
14. Belletti, F., Guidetti, M., Maiorano, A., Mantovani, F., Schifano, S.F., Tripiccione, R.: Monte Carlo Simulations of Spin Glass Systems on the Cell Broadband Engine. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part I. LNCS, vol. 6067, pp. 467–476. Springer, Heidelberg (2010)
15. Guidetti, M., Maiorano, A., Mantovani, F., Pivanti, M., Schifano, S.F., Tripiccione, R.: Monte Carlo Simulations of Spin Systems on Multi-core Processors. In: Jónasson, K. (ed.) PARA 2010, Part I. LNCS, vol. 7133, pp. 220–230. Springer, Heidelberg (2012)

7th Workshop on Virtualization in High-Performance Cloud Computing – VHPC2012

Michael Alexander¹, Gianluigi Zanetti², and Anastassios Nanos³

¹ TU Wien, Austria

² CRS4, Italy

³ NTUA, Greece

Virtualization has become a common abstraction layer in modern data centers, enabling resource owners to manage complex infrastructure independently of their applications. Conjointly, virtualization is becoming a driving technology for a manifold of industry grade IT services. The cloud concept includes the notion of a separation between resource owners and users, adding services such as hosted application frameworks and queueing. Utilizing the same infrastructure, clouds carry significant potential for use in high-performance scientific computing. The ability of clouds to provide for requests and releases of vast computing resources dynamically and close to the marginal cost of providing the services is unprecedented in the history of scientific and commercial computing.

Distributed computing concepts that leverage federated resource access are popular within the grid community, but have not seen previously desired deployed levels so far. Also, many of the scientific data centers are only beginning to apply virtualization and cloud concepts.

This year's workshop featured 5 submitted papers on diverse topics in HPC virtualization and 2 invited talks. Jakob Blomer from CERN presented CernVM, a distributed approach to high-throughput computing for the LHC using VMs. Vangelis Koukis from GRNET, presented and demonstrated Okeanos, the public cloud service for the Greek research and academic community.

The chairs would like to thank the Euro-Par and local organizers and the members of the program committee, along with the speakers and attendees, whose interaction contributed to a stimulating environment. VHPC is planning to continue the successful co-location with Euro-Par in 2013.

Pre-Copy and Post-Copy VM Live Migration for Memory Intensive Applications

Aidan Shribman¹ and Benoit Hudzia²

¹ SAP Research, Ra'anana, Israel

aidan.shribman@sap.com

² SAP Research, Belfast, UK

benoit.hudzia@sap.com

Abstract. Virtualization technology provides a means for server consolidation, reducing the number of physical servers required for running a given workload. Virtual Machine (VM) live migration facilitates the transfer of a running VM between physical hosts while appearing transparent to the running application. Memory intensive applications tend to obstruct the original pre-copy live migration process and may result in the failure of the migration process due to its inability to transfer memory faster than memory is dirtied by the running application. The focus of this paper is to present several techniques that can be applied to both pre-copy live migration and post-copy live migration to better support migration of memory intensive applications.

Keywords: Operating Systems, Hypervisors, Virtual Machines, Live Migration, Pre-Copy, Post-Copy, RDMA, QEMU, Linux/KVM.

1 Introduction

Virtualization is widely used in enterprise data centers as a means to reduce operational costs and increase operational flexibility. Adoption of this technology has surged following recent advances in x86 architecture such as multi-core and introduction of hardware assisted x86 extensions supporting full virtualization. Live migration is a core capability of modern hypervisors helping increase operational flexibility by serving as a foundation for capabilities such as High Availability (HA), Disaster Recovery (DR) and dynamic policy based migration of workloads for reducing power consumption in data center.

Live Migration Approaches. The live migration process handles the transfer of memory, CPU and hardware device state of the running VM. While the CPU and devices state are relatively small, typically in the size order of several KBs, size of memory may be in the order of many GBs. 1 GB of DRAM requires about 10 seconds to be transferred over a 1 GbE interconnect while 1 KB could be transferred over a 1 GbE in only 10 microseconds. Live migration schemes differ according to the order of state transfer: **Pre-Copy Live Migration:** Implemented in most hypervisors such as VMWare [4], Xen [1] and KVM

[2], presented by Brandford et al. [5] is both simple in design and does not require a fast interconnection between physical hosts. Memory is transferred to the destination host in a succession of iterations until the remaining dirty memory can be transferred in a short enough *stop and copy* phase which will not cause prolonged VM downtime. **Post-Copy Live Migration:** CPU and device state are transferred immediately to the destination host followed by transfer of execution control to the destination host. Source host memory is transferred in the background or fetched on-demand if needed by the running VM on the destination host. This migration scheme reduces the downtime and total migration time but incurs service degradation due to page faults which must be resolved over the network by the source host. **Hybrid Post-Copy Live Migration:** Adds a bounded pre-copy phase before entering the post-copy phase. The additional pre-copy phase reduces the number of future network bound page faults as a large portion of the VM memory is already pre-copied.

Live Migration Metrics. We evaluate the various live migration schemas by using the following metrics: **Downtime:** Is the amount of time in which the VM is suspended on the source host and until it becomes available again on the destination host. **Total Migration Time:** The overall time elapsed from the initiation of the live migration operation and until all resources on the source host are released. **Total Transferred Bytes:** The total number of bytes transferred over the network from the source host to the destination host. **Service Degradation:** The degree at which the VM's operation was slowed down due to the live migration process compared to uninterrupted VM execution.

Our Contribution. Is the design and implementation of several improvements to live migration to better support memory intensive applications:

Pre-Copy Live Migration: The fast memory dirtying rate of memory intensive applications relative to available network bandwidth challenges the pre-copy live migration scheme. In this paper we propose using a page reordering policy such that Least Recently Used (LRU) pages, with lower chances of being re-dirtied, are migrated earlier. Additionally, we propose using a fast delta encoder, Xor Binary Zero Run Length Encoding (XBZRLE), which reduces the cost of a page re-send.

Post-Copy Live Migration: Post-copy live migration has two major advantages over pre-copy live migration: (1) shorter downtime as only CPU and device state, several KB in magnitude, need to be migrated while the VM is stopped; (2) total migration time is short and deterministic as pages are not re-dirtied on the source host during live migration. We mitigate the challenge of service degradation by several means: (1) a Remote Direct Memory Access (RDMA) stack for low-latency resolution of network-bound page faults; (2) demand pre-paging (a form of pre-fetching) for reducing the overall number of page faults; (3) integration of page faulting mechanism with the Linux Memory Management Unit (MMU) so that only the thread waiting on the page fault is paused while other threads continue execution; (4) pre-copy post-copy hybrid live migration

scheme which helps further reduce the number of page faults when entering the post-copy phase.

2 Design and Implementation

This work introduces modifications to QEMU, originally developed by Fabrice Ballard [3] and KVM originally developed by Avi Kivity [2] on the Linux OS. Our enhancements are to be released as open source under GPL and LGPL licenses. A XBZRLE patch [8] was released during 2011 and is in the process of being integrated into the mainline QEMU code base.

2.1 Least Recently Used Page Reordering

Pre-copy page re-ordering can potentially reduce the total number of pages sent over the network by reducing the number of page re-sends. Checconi et al. [6] have proposed using LRU to dictate the order in which pages are transferred over the network to the destination host. In certain applicative workloads using such a policy may help reduce the page re-dirtying rate and hence reduce page re-sends. In turn downtime and total migration time are much reduced. In our implementation we propose subtle enhancements to the existing work:

QEMU constructs and maintains the LRU order according to when the VM pages were dirtied by the running application. Page dirty events are tracked by KVM in the Linux kernel using a dirty bitmap array. In each observation interval the dirty bitmap array indexed by page virtual address is cleared. The dirty bit is set for all pages dirtied in the observation interval. At the end of the observation interval there is no way of knowing what is the chronological order in which new pages were dirtied. Assuming address-based order (or any other order) would cause a bias in our LRU ordering we therefore propose using Fisher-Yates shuffle [18] to randomize dirty page order events before updating the LRU order.

As the pre-copy live migration scheme transfers memory in consecutive iterations we can use the LRU ordering not only to determine the order in which pages are sent within the interval but also for determining which pages need not be sent at all in the current interval as they will probably be re-dirtied again soon. Any pages at the tail of the LRU, namely the Most Recently Used (MRU) pages, are pages we do not send in the current interval. In our implementation we used an empiric value: the first 75 % LRU pages are transferred; the remainder 25 % MRU pages are not transferred. A more self-tuning approach for determining what is the percentage of pages to be sent in each interval may better suite real applicative workloads.

2.2 XBZRLE Delta Encoder

Reducing the number of page re-sends, via LRU page reordering, is speculative in nature and may have an adverse effect if future memory dirtying pattern

can't be derived from past memory dirtying pattern. A different approach to optimizing pre-copy live migration is to reduce the cost of each page re-send by using delta encoding for a compact representation of page updates. General purpose delta compression algorithms such as ZDelta are relatively slow reaching rates of about 50 MB/s [10]. Hudzia et al. [7] proposed the use of Xor Binary Run Length Encoding (XBRLE) for delta encoding of dirty pages for pre-copy live migration which reached rates of about 2 GB/s but had a relatively low compression ratio.

In this paper we present XBZRLE which provides a more compact encoding representation than XBRLE while additionally improving encoding speed. XBZRLE is comprised of two phases: (1) binary XOR for encoding the delta between the old memory page content and the new memory page content - XOR is both fast and is able of capturing the in-place memory changes - represented by the XOR as a sequence of non-zeros; (2) Zero Run Length Encoding (ZRLE) attempts to compress only the zero runs using Run Length Encoding (RLE) but does not attempt to encode non-zero runs. (as non-zero sequences have little chance for character repetition). Both phases XOR and ZRLE can utilize CPU Single Instruction Multiple Data (SIMD) instructions taking advance of multi-byte parallelism by using input and output characters of 8-bytes (=64 bit) rather than executing on byte sized characters. The XBZRLE delta encoder requires as input both the new page contents and the old page contents. The old page contents is cached each time a page is sent to the destination host. We currently use a rudimentary LRU cache but plan in future to use an advanced self-tuning cache such as ARC [11] or CAR [12] which would provide a higher hit-ratio without any increase in cache size. Cache size is determined by optimizing both cache hit-ratio and cache memory overhead incurred on host - for most workloads a 10 % of the VM memory size allocated for the cache - should provide ample hit-ratio.

In [9] we presented comparative results for both encoding speed and encoding size of various page encoders. We evaluated several encoders using 4 scenarios: sparse memory update pattern (step 1111 bytes alter 12 bytes), medium (step 701 alter 33), dense (step 203 alter 41) and very dense (step 121 alter 43). In order to compare compression algorithms candidates we first run a XOR phase (for getting a non compressed delta representation), following we run our candidates: LZO (xblzo), Google Snappy (xbsnappy), RLE (xbrie) and ZRLE (xbzrle). Results demonstrate that XBZRLE outperforms the other candidates in encoding speed achieving a 2200 MB/s compared 200 MB/s for byte-wise XBRLE, and 500 MB/s - 1200 MB/s for xblzo and xbsnappy. While fast - XBZRLE - does incur a 50% increase in compressed size compared with the slower xblzo and xbsnappy encoders.

2.3 Post-Copy Live Migration

In this subsection we present our implementation of post-copy live migration of VMs across an RDMA interconnect. While post-copy live migration is not new,

our QEMU and KVM based implementation has incorporated several innovations which make it especially capable of handling memory intensive applications.

Core to a high performing post-copy implementation is the efficient resolution of network bound page faults - which has also been a relative barrier to implementing post-copy live migration in general.

MMU Integration: Hirofuchi and Yamahata in Yabusame project [17] disclose post-copy enhancement of QEMU/KVM using a character device and page faulting transport via user mode. Our approach was to extend the Linux MMU such that network bound page faults are handled directly in the kernel much like swap in of page from a disk-based swap device incurring no context switches into user mode. By introducing a new flag in the *page table entries* we indicate that the page is backed by memory residing on a remote host. While Yabusame causes a full VM freeze during remote page fault - our implementation only causes the requesting thread to pause while all other VM threads continue uninterrupted.

RDMA Interconnects: The concept of remote DRAM backed swap was proposed by Comer and Griffioen [15], however, only the recent advancement in network interconnects performance and accompanying standardization in the RDMA stack have enabled to unleash its power. Especially notable is OFED and its promotion of standard RDMA semantics for all underlying hardware and software stacks such as InfiniBand, IWARP and RDMA over Converged Ethernet (RoCE). While 40 Gbps InfiniBand and 10 Gbps Ethernet provide excellent performance it is also possible to use a pure software stack such as softIWARP atop 1 GbE making post-copy live migration feasible in many cost-conscious environments without the need for up-front hardware expenses. While RDMA is an enabler to post-copy live migration it can also benefit pre-copy live migration as presented by Huang et al. [13].

Demand Pre-paging: Was first presented by Kaplan et al. [14] who demonstrated that by altering the amount of pre-fetched pages according to the load characteristics reduced the percentage of page faults in specific scenarios while not incurring increased service degradation in other scenarios. By adding a simple pre-fetcher that demands the 40 pages surrounding the faulted page we managed to reduce service degradation. Hines et al [16] demonstrated the effectiveness of pre-paging for reducing network-bound page faults to be within 21% of the VM working set size.

Hybrid Post-Copy: We additionally implemented hybrid post-copy allowing for a pre-defined bound pre-copy phase to proceed the post-copy phase as a means for reducing the number of future page faults.

3 Evaluation

While initially considering realistic applications such as the SAP Sales & Distribution Benchmark as used by Hudzia et al [7], it became apparent that these may vary in behavior greatly between runs thus making comparative testing

of live migration schemes difficult. We thus focused on artificial benchmarks which may not fully represent real memory intensive applications but are highly deterministic in behavior.

Pre-Copy Evaluation. We built a synthetic multi threaded workload generator *appmembench*, which dirties different groups of pages in different frequencies using 8 threads, a working set of 512 MB, and a workload characterized by a memory dirtying rate of 1 GBps (adding the full 4 KB page size as dirty memory for each page partially dirtied). Workload was run on a guest VM of: 4 x vCPU; 1 GB RAM; Each physical host with 2 cores, 8 GB RAM, while hosts were interconnected with 1 Gbps Ethernet network, we limited network bandwidth to only 30 MB/s or 240 Mbps to maintain a high *application-dirty-rate / network-transfer-rate* ratio - characteristic to real memory intensive workloads.

Figure 1 compares three pre-copy live migration implementations: (1) the original QEMU implementation - 'original'; (2) Page LRU reordering modification - 'prio'; (3) XBZRLE modification - 'xbzrle'. The chronological sequence of events presented in the chart: 0 seconds - start running appmembench; 30 seconds - start live migration; 150 seconds - force migration (by setting a high max downtime value); 180 seconds - end of measurements.

The original live migration incurs a short downtime right after live migration initiation at 30 seconds, continues to work without any degraded service, but does not manage to complete the migration and transfers control only after 150 seconds and in the process sustaining a 6 seconds downtime. XBZRLE causes a severe service degradation - indicating that the application and XBZRLE are competing on limited compute resources - but it succeeds in completing the live migration after 105 seconds. (Although due to degraded service level XBZRLE had to cope with a much lower rate of page dirtying than the original live migration which did not cause service degradation). Finally LRU page reordering caused a 50% service degradation, also due to CPU overcommit, and only migrated after 150 seconds but has a slightly shorter downtime of about 4 seconds compared with the original implementation.

The appmembench benchmark ran 8 threads on 4 vCPUs atop 2 physical CPU cores, causing severe CPU pressure. As both XBZRLE and LRU page reordering use compute to mitigate network bandwidth strain the result in our case was an adverse effect on overall system performance due to competition between the running VM and the live migration process itself. Also highlighted by this test is the absence in our PoC of an automatic disabling mechanisms during CPU pressure.

Post-Copy versus Pre-Copy. So as to highlight post-copy's ability to cope with very high workloads we use Google Stress Application Test (SAT) - a scalable high-performance benchmark. Google SAT performs large bit-wise inverse operations which may not fully represent the types of operations done by real applications but creates a high multi-threaded workload which can saturate physical resources. For our tests we used the following guest VM specification: 2 x vCPU ; 1, 2, 4, 8, and 16 GB of memory; Google SAT benchmark with 1 GB working set; 1 Gbps Ethernet Network between hosts.

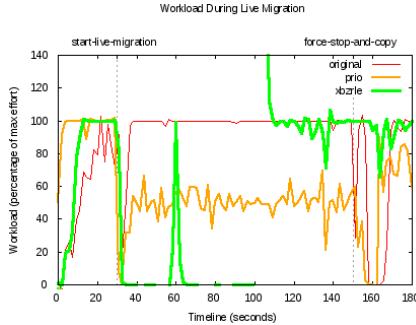
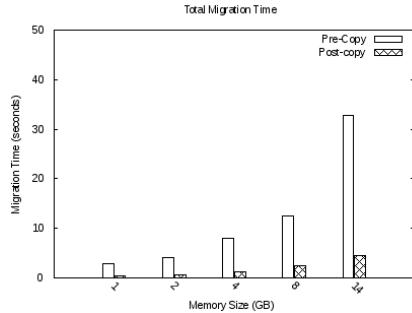
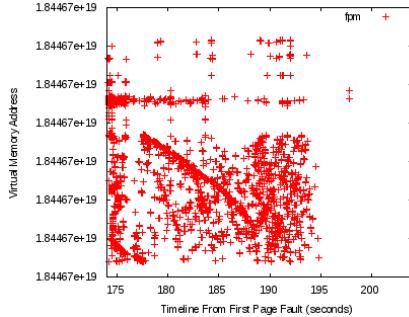
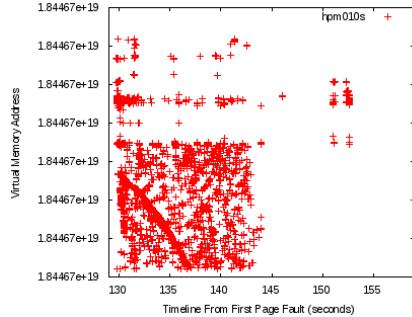
**Fig. 1.** Pre-Copy Evaluation**Fig. 2.** Post-Copy vs. Pre-Copy

Figure 2 depicts total migration time for both pre-copy original implementation and post-copy live migration for various VM sizes. Post-copy live migration completed the live migration in about 10% of the time required for the original pre-copy implementation, indicating that the original pre-copy sends page updates over and over again in average 10 times each compared to post-copy which sends each memory page exactly once.

**Fig. 3.** Full Post-Copy**Fig. 4.** Hybrid Post-Copy

Hybrid Post-Copy Evaluation. We used the following guest VM specification: 2 x vCPU ; 4 GB of memory; 1 GB Google SAT working set; 1 Gbps Ethernet Network between hosts. Network bound page faults are the main cause of degraded service right after transfer of control to the destination host and therefore their pattern and timespan can help assess the severity of this period. In figures 3, 4, individual network bound page fault events were recorded: Using virtual address as y coordinate and using relative time from first page fault as x coordinate. The figures depict that the pre-copy phase helps reducing page faults induced application service degradation - as can be seen when comparing full post-copy figure 3 versus hybrid post-copy 10s (a 10 second pre-copy phase)

figure 4. The contribution of the pre-copy phase is nonetheless finite and does not go beyond a certain point as was determined when comparing our results for the hybrid post-copy 10s case to hybrid post-copy 20s and hybrid post-copy 40s.

4 Conclusions and Future Work

Migrating large memory intensive workloads challenges the existing live migration implementations. In this work we presented our enhancements and optimization over the original pre-copy live migration KVM QEMU implementation. We presented XBZRLE and LRU page reordering which support live migration of applications with high memory dirtying rate relative to available network bandwidth. Next we disclosed our enhanced post-copy live migration implementation which by subsuming demand pre-paging, fast RDMA interconnects, MMU integration and hybrid post-copy can provide a fast and predictable live-migration over fast interconnects maintaining minimal page fault incurred service degradation. In the future we plan extending our work on live migration to handle transfer of CPU and memory state between a cluster of hosts serving as a single **Virtual Distributed Shared Memory** system.

Acknowledgments. This work was funded by grant from the Invest Northern Ireland regional economic development agency covering work done by John Stuart, David Leib and Benoit Hudzia. Additional funding supported by the EU FP7 project VISION Cloud covering Aidan Shribman's work. We thank Chaim Bendelac and Eliezer Levy for their invaluable feedback.

References

1. 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: Proceedings of the 19th ACM Symposium on Operating Systems Principles, pp. 164–177. ACM, New York (2003)
2. Kivity, A.: KVM: Kernel-based Virtualization Machine, <http://www.linux-kvm.org>
3. Ballard, F.: QEMU: Open Source Processor Emulator, <http://wiki.qemu.org>
4. VMWare Inc.: VMWare, <http://www.vmware.com>
5. Bradford, R., Kotsovios, E., Feldmann, A., Schiberg, H.: Live wide-area migration of virtual machines including local persistent state. In: VEE 2007 Proceedings of the 3rd International Conference on Virtual Execution Environments, pp. 169–179 (2007)
6. Checconi, F., Cucinotta, T., Stein, M.: Real-Time Issues in Live Migration of Virtual Machines. In: Lin, H.-X., Alexander, M., Forsell, M., Knüpfer, A., Prodan, R., Sousa, L., Streit, A. (eds.) Euro-Par 2009. LNCS, vol. 6043, pp. 454–466. Springer, Heidelberg (2010)
7. Hudzia, B., Svard, P., Tordsson, J., Elmroth, E.: Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines. In: VEE 2011 Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 111–120 (2011)

8. Shribman, A., Hudzia, B., Svard, P.: PATCH v4: XBZRLE delta for live migration of large memory apps, <http://patchwork.ozlabs.org/patch/108868/>
9. Shribman, A., Hudzia, B., Svard, P.: Microbenchmarks of page delta encoders, <http://lists.gnu.org/archive/html/qemu-devel/2011-08/msg00214.html>
10. Memon, N., Suel, T.: The zdelta-2.0 experimental results, <http://cis.poly.edu/zdelta/results.shtml>
11. Megiddo, N., Modha, D.S.: ARC: A Self-Tuning, Low Overhead Replacement Cache. In: FAST 2003 Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pp. 115–130 (2003)
12. Bansal, S., Modha, D.S.: CAR: Clock with Adaptive Replacement. In: FAST 2004 Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp. 187–200 (2004)
13. Huang, W., Gao, Q., Liu, J., Panda, D.K.: High performance virtual machine migration with RDMA over modern interconnects. In: CLUSTER 2007 Proceedings of the 2007 IEEE International Conference on Cluster Computing, pp. 11–20 (2007)
14. Kaplan, S.F., McGeoch, L.A., Cole, M.F.: Adaptive Caching for Demand Prepaging. In: ISMM 2002 Proceedings of the 3rd International Symposium on Memory Management, pp. 114–126 (2002)
15. Comer, D.E., Griffioen, J.: A New Design for Distributed Systems: The Remote Memory Model, <http://docs.lib.psu.edu/cstech/830>
16. Hines, M.R., Gopalan, K.: Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning. In: VEE 2009 Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 51–60 (2009)
17. Hirofuchi, T., Yamahata, I.: Yabusame: Postcopy Live Migration for QEMU/KVM. In: KVM Forum 2011 (2011)
18. Fisher, R.A., Yates, F.: Statistical tables for biological, agricultural and medical research, 3rd edn., pp. 26–27. Oliver & Boyd, London (1938, 1948), OCLC 14222135

Xen2MX: Towards High-Performance Communication in the Cloud

Anastassios Nanos and Nectarios Koziris

Computing Systems Laboratory,
National Technical University of Athens,
`{ananos,nkoziris}@cslab.ece.ntua.gr`

Abstract. Efficient VM communication in Cloud computing infrastructures is an important aspect of HPC application deployment in clusters of VMs. In this paper we present Xen2MX, a high-performance messaging protocol, binary compatible with Myrinet/MX and wire compatible with MXoE. Its design is based on MX and its port over generic Ethernet adapters, Open-MX. Xen2MX combines the zero-copy characteristics of Open-MX with Xen's memory sharing techniques, in order to construct the most efficient data path for high-performance communication, achievable with software techniques. Using Xen2MX, we are able to reduce the round-trip latency to $14\mu s$, compared to directly attached devices ($13\mu s$) and to a software bridge setup ($44\mu s$).

1 Introduction

Cloud computing infrastructures offer dedicated execution, isolation and flexibility to a vast number of services, providing huge processing power; this feature makes them ideal for mass deployment of compute-intensive applications. However, I/O-intensive applications suffer from poor performance in the cloud context. Numerous approaches have been proposed to alter the programming paradigm of the Cloud [1,2] leading to a less communication oriented model, though decentralized and distributed.

During the last decade, I/O Virtualization (IOV) techniques have been introduced to provide near-native performance both in 10GbE and exotic interconnection frameworks [3,4]. Compared to the common case of Virtual Machine (VM) communication (software bridges or routing techniques), IOV has managed to overcome specific limitations in terms of performance. The community has proposed several optimizations to IOV, but still, a major issue correlated with its design remains unsolved: scalability. The number of VMs that enjoy direct data-paths to the network is limited by the hardware capabilities of the specific IOV-enabled adapter. In the era of multi/many-cores, where VM containers would be able to host a great number of VMs, will IOV adapters be able to cope with servicing requests from the system or the network at a sufficient rate ? this problem is multi-parametrical. We believe that close investigation to virtualization system approaches is necessary, using software tools that can mark out efficient data paths and sources of overhead that need to be eliminated.

As we move towards the standardization of Ethernet in both worlds, Cloud computing and High-Performance Computing (HPC), we need a way to study the effect of message-passing protocols in the Cloud, without having to suffer from TCP/IP’s complexity. However, current approaches do not provide a software solution to efficiently exploit hypervisor abstractions to access hardware. In previous studies [5,6,7] we have attempted to examine the trade-offs related to device sharing using custom lower-level protocols and simple microbenchmarks. We move forward to a more generic design, in order to understand and optimize the way VMs interact with the network in an HPC context.

In this work we describe the design of Xen2MX, a high-performance interconnection protocol for virtualized environments. Xen2MX is binary compatible with MX and wire compatible to MXoE, the Ethernet mode of Myrinet’s MX protocol. Although our prototype implementation is in early stages, results from the original MX benchmarks over Xen2MX are promising, reducing the round-trip latency to as low as $14\mu s$ compared to $44\mu s$ of a software bridge setup, and $13\mu s$ of the IOV case. We juxtapose our findings with IOV techniques and examine possible ways to optimize our prototype in order to achieve near-native performance.

2 Motivation and Background

Overview of the Xen Architecture. Xen [8] is a popular Virtual Machine Monitor (VMM) that supports Paravirtualization (PV). Data access is handled by privileged guests called *driver domains* that help VMs interact with the hardware, based on the *split driver model*. With Single Root I/O Virtualization (SR-IOV) [3], VMs exchange data with the network via a direct VM-to-NIC data path provided by a combination of hardware and software techniques: smart adapters export multiple PCI functions to the Virtual Machine Monitor (VMM); hypervisors assign these functions to VMs, so guest kernels run native drivers and control part of the adapter’s capabilities.

In Xen, memory is virtualized in order to provide contiguous regions to OS’s running on guest domains. This is achieved by adding a per-domain memory abstraction called *pseudo-physical* memory. So, in Xen, *machine memory* refers to the physical memory of the entire system whereas *pseudo-physical* memory refers to the physical memory that the OS in any guest domain is aware of.

To efficiently share pages across guest domains, Xen exports a *grant* mechanism. Xen’s grants are stored in *grant tables* and provide a generic mechanism for memory sharing between domains. Two guests can initialize an *event channel* and exchange events that trigger the execution of the corresponding handlers. This mechanism is often used along with I/O rings, when one guest wants to inform another about the placement of a new request or response in the common ring.

Xen’s PV network architecture is based on the split driver model. Guest VMs host the *netfront* driver, which exports a generic Ethernet API to the guest’s kernel-space. The driver domain hosts the hardware specific driver and the

netback driver, which interacts with the frontend using the event channel mechanism and injects frames to the NIC via a software bridge.

Design Choices for a Message-Passing Protocol. In setting out to integrate high-performance communication semantics in a virtual environment, we have to respect certain principles: the result of our effort must be portable, simple, scalable, and robust. It must also achieve high-performance in terms of both low-latency communication and line-rate bandwidth. Below, we briefly elaborate on each of these design goals:

Simple and Portable: It is rare for system developers to possess the required resources to port their code on various interconnection protocols. It seems that abiding by feature-specific APIs slows down the adoption of new and improved technology. So the obvious choice for them is a widely adopted protocol with a well specified programming interface. The resulting design and code must be simple, understandable and tailored to virtualization demands.

Scalable: It is common case in the HPC world to associate send/receive buffers to any form of communication endpoints (connection oriented or not). This has been widely considered as a bad move [9], especially due to the lack of scalability this implies. As the number of nodes/hosts climb the scale of 1000, then buffers associated with each node are non-negligible in terms of space requirements and management – let alone the mapping of every single host of the network. This seems like a waste of resources compared to just spending a few CPU cycles to poll; it is a fairly significant trade-off we are obliged to take in terms of scalability.

Robust: One of MPI’s initial goals was to be kept simple. This is the main reason that MPI’s fault tolerance is not elaborate. As a result, the lower-layer interconnection protocol must ensure that error conditions can be handled properly. Connection oriented semantics can solve this issue, keeping the protocol semantics as closely coupled to MPI as possible.

Interconnection Protocol. Many custom programming interfaces have been proposed for existing message passing stacks. However, MPI is the current standard for communication on the scientific applications front. For instance, a message passing layer does not need to implement collective communication from scratch – it is only necessary to build the interconnect abstraction of MPI, the Byte Transfer Layer (BTL), that translates MPI semantics into actual calls for the interconnect to handle. One popular lower-level protocol that acts as a BTL for all popular MPI implementations and supports all necessary communication capabilities used by MPI is Myrinet eXpress (MX [10]).

Communication-sensitive applications may obtain significant performance improvement due to dedicated high-speed network technologies. However, as most applications do not yet exhibit communication bottlenecks, many clusters still use commodity networking technologies such as gigabit Ethernet. Indeed, this category of parallel applications is often bound to an implementation of MPI over TCP/IP; nevertheless, TCP/IP has often been criticized as being slow in the context of high-performance computing. As a result, there is a new trend

in the HPC market: porting/building high-performance protocols over generic Ethernet.

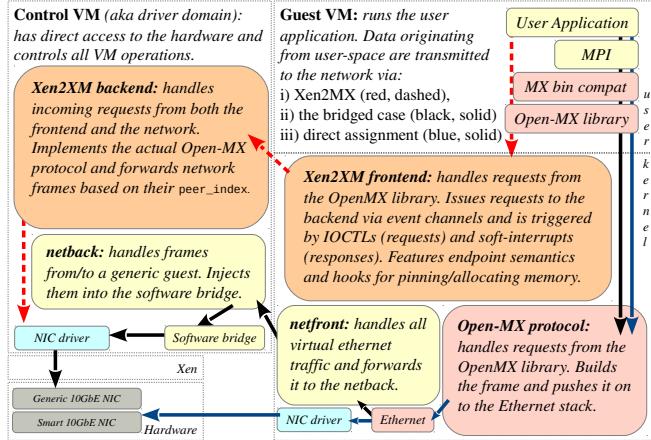
Open-MX [11] is the software implementation of the MX protocol over generic Ethernet adapters. MX employs user-level networking techniques to achieve high-performance communication. It exploits the capabilities of Myrinet and Myri-10G hardware at the application level while providing low-latency and high bandwidth (less than $2\mu\text{s}$ and 1250MB/s data rate). All the actual communication management is implemented in the user-space library and in the firmware. Open-MX follows the same implementation model and semantics as MX; their main difference is that OS-bypass is not possible with generic adapters. Open-MX handles messages the same way as MX, depending on the size of the requested transfer. Its vital building blocks are: *Endpoints*: An endpoint can be considered as a virtualized instance of a device, a logical source or destination of all communications in high-performance interconnects. *Events*: Applications interact with Open-MX through events, a scalable method of communication between kernel-space and user-space. Events may represent receive notifications or send completions. *Regions*: Memory regions are sets of memory *segments* that contain virtually contiguous memory areas allocated by the application. Regions can be the source or the destination of a message and are mainly used in the rendez-vous communication scenario.

When it comes to cloud computing, and specifically virtualization, things get a bit more complicated: virtualized environments are hostile territory for communication-intensive scientific applications. The extra layers of abstraction that comprise the intermediate virtualization layers reduce the overall performance significantly. IOV enabled devices install a direct application-to-hardware data path giving the specific VM the necessary network capacity. Nevertheless, this approach complicates the setup in a way that hardware limitations arise early on the scaling factor and important virtualization features (migration, checkpointing, flexibility in general) are not fully exploited.

3 Xen2MX: Design and Implementation

Our approach proposes the integration of the split driver model described above, to Open-MX. Using endpoints, the asynchronous event mechanism and smart page mappings, VM's user-space is able to communicate with the network using the MX protocol without suffering the overhead of the software Ethernet bridge, while at the same time, the driver domain keeps full control of the network flow. This is actually the core idea of Xen2MX¹: applications running on VM user-space interact with the MX library, using the MX binary compatibility Open-MX offers; the library makes calls to the frontend module, keeping the protocol semantics intact; the frontend forwards requests to the backend module and vice-versa, depending on the direction of data flow; finally, frames are split and distributed to the frontend's pages or built based on the message size and are transmitted to the network via Linux kernel's Ethernet layers (see Figure 1).

¹ <https://github.com/ananos/xen2mx>

**Fig. 1.** Xen2MX

The main communication mechanism between the frontend and the backend is Xen’s event channels and the grant mechanism. Memory registration is synchronous, providing the backend with guest user-allocated memory pages. These pages can be attached to socket buffers as fragments, forming a packet to be transmitted to the network. Otherwise, these pages are the destination of a receive request and get populated with data originating from the network.

Endpoints feature send and receive queues, a statically allocated buffer that acts as the source or destination of MEDIUM sends and receives (<64KB). These queues, along with their buffers, are mapped in the backend, granted using Xen’s mechanisms and addressed by lower-level Open-MX layers.

Frontend–Backend communication: The basic block of our design is how the guest interacts with the driver domain. Xen offers basic communication methods for consumer–producer schemes, on top of which we implement a notification mechanism using both soft interrupts and polling, depending on system demands.

Control data exchange is realized using two cyclic rings, shared between both the backend and the frontend. The first ring handles management commands (requests) and most of the send path. The second ring does completely the opposite: it handles receive notifications. For simplicity, SMALL message data is copied across these rings. This feature, combined with polling on both ends, gives the lowest achievable latency in this setup.

Data exchange for MEDIUM and LARGE messages is realized using a more complicated scenario. As in the Open-MX case, MEDIUM messages use the send and receive queues; their buffers are static, allocated upon endpoint initialization and are addressed using internal indexing. LARGE messages use the registered space (memory regions, see Section 2).

Regions in Open-MX are allocated in user-space and registered using a specific IOCTL. In Xen2MX, the frontend grants the region space to the backend, segment by segment. In the frontend, each segment contains extra fields that keep track of granted pages, which are released only when the backend has finished with them.

The same stands for the send and receive queue buffers. The pages that comprise these queues are granted by the frontend to the backend, resulting in consistent, identical queues addressable by the backend.

The tricky part on this approach is that there is a lot of communication in order to keep regions and queues consistent between the frontend and the backend. The overhead imposed by this communication is non-negligible; therefore, in order to achieve the desirable bandwidth, we need to finetune the way data flow from/to the network. Specifically:

SMALL messages are being copied across, so there's not much we can do about it. We keep the number of copies as low as the normal Open-MX, suffering only from the constant overhead of the message being transmitted to the backend.

MEDIUM messages are being sent/received via the relevant queues. The situation here is the same as in **SMALL** messages. The added overhead of pinning and granting the send/receive queues is only relevant to the opening/allocation of the endpoint structures which does not occur on the critical path.

LARGE messages are send/received via memory regions. However, memory registration is a time-consuming part of the whole process, that needs to be closely analyzed in order to understand the sources of added overhead and overcome them.

On the guest's front, on top of the original allocation and pinning process, we loop around each region segment and grant all pages to the driver domain, keeping track of the relevant grant references (which are actually 32-bit unsigned positive integers). We store these integers to a number of reference pages, which we grant to the backend. At first, the backend accepts these pages and dereferences their content to obtain the grant references of the original pages. This approach is different than the one used both in Xen's blkback and netback drivers, due to the large numbers of references we export.

Peer/Neighbor discovery: Contemporary message passing protocols, prior to communication, discover the set of network nodes in order to establish the map of the network. When a node comes up in Open-MX, it advertises its existence using *raw endpoints*, a stripped down version of the original endpoint instance. This information is saved in each node's **peer table**, from which any application that wants to communicate with the network gets its **peer index**. This information is essential for upper-layer protocols (such as MPI) which use hostnames to establish initial communication over TCP/IP with their peers.

In our design, peer discovery appears to be more complicated: all VMs that co-exist in the same VM container, share the Ethernet interface(s) of the host machine. Therefore, the network mapping would be incomplete, since multiple peers could not be added to the table using the same interface identification number. To overcome this, we extended the peer table to support multiple peers attached to the same interface.

VM to VM Communication: A common approach adopted by IaaS providers is that the user is not directly aware of the physical placement of the ordered VMs.

As a result, the user may end up (by chance) getting his communication-intensive HPC application executed in VMs that co-exist in the same physical machine. If this is the case, then IOV techniques fail dramatically: for a VM to communicate with its peer, data have to flow from the VM’s memory to the hardware and then back to the peer VM’s memory. This unfortunate situation is not going to happen if we use the split driver model. In Xen2MX, the backend can check the `peer_index` of the destination endpoint, and if it finds it in the local peer table, shared memory communication gets triggered and data get propagated to the peer VM automagically.

4 Preliminary Evaluation

In Xen2MX, the user-space library as well as the API are intact. As a result, we use a native MX microbenchmark, `mx_pingpong`, to measure round-trip latency and ping-pong bandwidth between one guest using our implementation and a native linux box, running the latest version of Open-MX². Both machines are identical, featuring one Quad core Intel Xeon 2.4GHz with an Intel 5500 chipset and 4GB of main memory. The network adapters used are two Myricom 10G-PCIE-8A 10GbE NICs in Ethernet mode, connected back-to-back. We deployed our prototype using Xen version 4.2-unstable³ and Linux kernel version 3.4.0.

We use three different cases to setup our experiment. *Bridged* is the default Xen approach, using a software bridge, *PCI-attached* is the case where the physical device is attached directly to the VM, using Xen’s *pass-through* mechanism. *Xen2MX* is our approach.

Figure 2 plots the round-trip latency achieved under the three aforementioned setups. Our approach is almost identical to *PCI-attached* for SMALL messages whereas the *Bridged* case achieves $44\mu s$. Using Xen2MX, transferring 1 byte across takes $14\mu s$, less than half of the *Bridged* case and approximately $1\mu s$ more than the best measured (*PCI-attached* case). Figure 3 shows the ping-pong bandwidth measured on the three setups. Our approach outperforms the *Bridged* case which exhibits unstable behavior. After 512KB, performance drops to as low as 10 MB/s, for reasons that we could not explain. We attribute this issue, to the fact that the netback / netfront drivers are not optimized towards high-performance communication; hence this misbehavior when being overwhelmed with buffer handling at this rate.

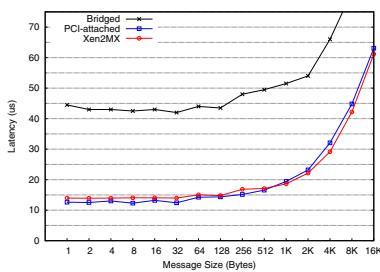


Fig. 2. Latency (lower is better)

² Open-MX 1.5.2

³ Changeset 25099:4bd752a4cdf3

Our approach seems to follow the scaling of the PCI-attached case up until a specific point (32KB). This point is where memory registration becomes noticeable; memory registration is not always on the critical path, depending on the implementation of the protocol. Rendez-vous semantics, present in MX and Open-MX message exchange, oblige `mx_pingpong` to register all the memory needed for communication on demand. As a result, the part where the two peers agree on the communication parameters and register memory regions is included in the measurement. We would like to keep the protocol intact, as this is considered the common case in MPI execution, so we are in the process of optimizing the way our initial implementation handles registration and granting.

5 Conclusion and Future Work

We have presented the design and a prototype implementation of Xen2MX, a software port of the MX protocol to the Xen split driver model, based on Open-MX. Xen2MX benefits from all Open-MX's features to provide binary compatibility with MX as well as wire compatibility with MXoE. Our initial prototype is able to achieve low-latency compared to Ethernet software bridges, the generic case of networking in virtualized environments, as well as comparable results to IOV techniques. Our design is applicable to all paravirtualized approaches and provides an easy, scalable way to deploy communication-intensive applications in the Cloud.

We aspire to perform a detailed analysis of all overheads imposed by virtualization layers in order to optimize our initial implementation. Our future agenda consists of exploiting Xen2MX's binary compatibility with MX (and thus MPI), to deploy HPC applications over this framework and benefit from the flexibility of the split driver design and the scalability of using more than one driver domains. We also plan to closely follow the trends of Ethernet-based message passing protocols such as the Common Communication Interface (CCI [9]) and provide an easy way to adapt our framework to these protocols.

Acknowledgements. This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund. We would also like to thank the anonymous reviewers for their invaluable feedback.

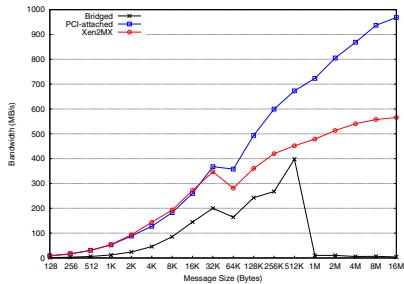


Fig. 3. Ping-pong bandwidth

References

1. Murray, D.G., Hand, S.: Scripting the cloud with skywriting. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud 2010, p. 12. USENIX Association, Berkeley (2010)
2. Murray, D.G., Schwarzkopf, M., Smowton, C., Smith, S., Madhavapeddy, A., Hand, S.: CIEL: a universal execution engine for distributed data-flow computing. In: NSDI 2011, p. 9. USENIX Association, Berkeley (2011)
3. PCI SIG: SR-IOV (2007),
http://www.pcisig.com/specifications/iov/single_root/
4. Liu, J., Huang, W., Abali, B., Panda, D.K.: High performance VMM-bypass I/O in virtual machines. In: ATEC 2006: Proc. of the USENIX 2006 Annual Technical Conference, p. 3. USENIX, Berkeley (2006)
5. Nanos, A., Koziris, N.: MyriXen: Message Passing in Xen Virtual Machines over Myrinet and Ethernet. In: 4th Workshop on Virtualization in High-Performance Cloud Computing, The Netherlands (2009)
6. Nanos, A., Goumas, G., Koziris, N.: Exploring I/O Virtualization Data paths for MPI Applications in a Cluster of VMs: A Networking Perspective. In: VHPC 2010, Naples-Ischia, Italy (2010)
7. Nanos, A., Nikoleris, N., Psomadakis, S., Kozyri, E., Koziris, N.: A Smart HPC Interconnect for Clusters of Virtual Machines. In: VHPC 2011, France (2011)
8. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I.A., Warfield, A.: Xen and the Art of Virtualization. In: SOSP 2003: Proc. of the 19th ACM Symposium on Operating Systems Principles, pp. 164–177. ACM, NY (2003)
9. Atchley, S., Dillow, D., Shipman, G.M., Geoffray, P., Squyres, J.M., Bosilca, G., Minnich, R.: The Common Communication Interface (CCI). In: IEEE 19th Annual Symposium on High Performance Interconnects, HOTI 2011, Santa Clara, CA, USA, August 24-26 (2011)
10. Myricom: Myrinet eXpress (MX): A High Performance, Low-Level, Message-Passing Interface for Myrinet (2006), <http://www.myri.com/scs/MX/doc/mx.pdf>
11. Goglin, B.: High-Performance Message Passing over generic Ethernet Hardware with Open-MX. Parallel Computing 37(2), 85–100 (2011), Open-MX

Themis: Energy Efficient Management of Workloads in Virtualized Data Centers

Gaurav Dhiman¹, Vasileios Kontorinis¹, Raid Ayoub¹, Liuyi Zhang¹, Chris Sadler²,
Dean Tullsen¹, and Tajana Simunic Rosing^{1,*}

¹ UCSD

² Google

tajana@ucsd.edu

Abstract. Virtualized data centers facilitate higher resource utilization and energy efficiency through consolidation. However, mixing services-oriented workloads with throughput (batch) jobs is typically avoided due to complex interactions and widely different quality of service (QoS) requirements. We introduce a complete VM resource management framework, called Themis, which manages combined services and batch jobs, maximizing energy-efficient throughput of the latter without sacrificing the service guarantees of the former. Themis' resource management policy outperforms the prior proposed policies by up to 35% on average in work done per Joule when measured on a data center testbed.

1 Introduction

Virtualization has rapidly gained prominence in modern data center deployments, since it provides better fault isolation, improved system manageability, and reduced operational cost through resource consolidation and migration [8]. It is common for a data center to host a mix of interactive service-oriented and throughput-oriented batch jobs. These two types of jobs are usually partitioned into separate sections of the data center [15] because we lack a mechanism for managing their diverse performance requirements. Services typically have strict response time guarantees and the cost of violating those agreements is high [15]. Batch jobs often have long-term performance targets, where immediate response is not vital.

A number of systems for VM management have been proposed in the past. Eucalyptus [23], OpenNebula [25], OpenStack [31] and Usher [18] are open source systems, which include support for managing VM creation and allocation across a cluster, and provide API for migration. However, these solutions generally do not have online VM scheduling policies to dynamically consolidate or redistribute VMs, but instead focus on the initial resource allocation and assignment of VMs to physical machines. In [29], the authors propose a system which dynamically schedules the VMs based on their CPU, memory and network utilization to improve the overall performance. In [6] the authors propose dynamic consolidation and redistribution of VMs for managing QoS requirements of different service VMs in the cluster. Entropy

* Corresponding author.

[14] uses constraint programming to determine a globally optimal solution for VM scheduling in contrast to the first fit decreasing heuristic used by [6, 29], which can result in globally sub-optimal placement of VMs. However, these approaches are not QoS aware. They assume that the CPU utilization adds upon VM consolidation, which is not true for heterogeneous VM consolidation.

Management of QoS within operating systems for latency sensitive applications in a heterogeneous workload mix running on standalone systems has been studied in Stanford SMART scheduler [23] and QLinux [28] projects. They ensure timely access to the CPU for the latency sensitive applications while maintaining proportional sharing of CPU for the batch jobs. Similar solutions have been proposed for virtualized environments as well [17, 24]. We show that the software level support for QoS through timely CPU access is not sufficient to guarantee QoS in presence of interference effects in modern multi-core based systems.

The interference effects due to shared resource usage by co-scheduled workloads on modern multi-core based platforms has been studied before at both the OS and hypervisor levels. The work in [5, 9] explores the problem exclusively for batch jobs. The research in [9, 20] takes the same problem of shared resource usage to the cluster level using virtualization, again just for batch workloads. In [21, 26], the authors use CPU capping to ensure QoS requirements are met. However, their focus is either on batch workloads [21] or services [26], but not a mix of the two.

State of the art techniques that focus on consolidating homogeneous workloads do not scale well for resource management with heterogeneous workloads. To solve this problem we designed Themis, a system for VM resource management in virtualized clusters. Themis includes a monitoring infrastructure which allows it to actively measure both performance and QoS metrics of different types of workloads within the data center. It is built as extension to Xen [3], and as such could be integrated into any open source system which can leverage Xen. We evaluate Themis on a testbed built of state-of-the-art servers with workloads representative of both services and batch jobs. Our evaluation shows that Themis can improve energy efficiency by up to 35% over the best proposed policies for resource management in consolidated environments while meeting both service and batch job performance targets.

2 Themis Design

In this section we provide details of the design and implementation of our system, Themis, for managing diverse workloads in the data center. We classify the workloads into two categories [15]: **(1) Services.** The primary goal for these workloads is to serve the user request within a given time bound to maintain a QoS level. In this paper we use RUBiS [1] as representative of services. Rubis is a multi-tier online service that implements the core functions of an auction site including selling, browsing, and bidding. It contains a front-end Apache PHP web server and a back-end MySQL database. RUBiS provides workloads of different mixes for the client sessions that are emulated on a separate machine. We use the ‘browsing mix’, which emulates a web-intensive user browsing experience. **(2) Batch jobs.** These workloads refer to resource intensive jobs that are representative of the analytics, number

crunching, and scientific computing class of workloads. The primary goal of these jobs is to maximize the overall instruction throughput, with no specific response time requirements. Since the goal of this paper is to study the effects of CPU and memory interference on the consolidation of different types of data center workloads, we use representative workloads from SPEC2000 and PARSEC benchmark suites, as our batch workloads. These workloads can be used to approximate memory and CPU intensive phases of typical data center batch jobs such as MapReduce [30].

Themis' objectives are to: (1) satisfy the QoS requirement of the service jobs; (2) maximize the batch job throughput; (3) minimize the power consumption. We define a new metric to capture these goals. It measures the batch job throughput/Watt, multiplied by a factor q that reflects how closely services QoS requirements are met. If QoS requirements are *not* met, then q is set to zero. Maximizing qMIPS/Watt implies that it is acceptable to sacrifice a bit of performance of service jobs as long as we still meet the strict service level agreements, increase batch throughput, and increase the overall energy-efficiency of the system.

$$q\text{MIPS}/\text{Watt} = q * (\text{batchjobMIPS})/\text{SystemWatts} \quad (1)$$

2.1 Motivation for Themis

Many of the modern hypervisor schedulers are based on proportional sharing of CPU resources. Xen uses a credit scheduler, where each VM's proportional share is specified through ‘weight’. Based on weight, the physical CPU resources (or credits) are distributed to the virtual CPUs (vCPUs) in proportion of their weight, with vCPU priority recalculated based on the credits the VM has. There are three priority levels: (1) ‘Over’: the lowest priority that a VM is set to when it exhausts its credits; (2) ‘Under’: medium priority; and (3) ‘Boost’: the highest priority used for low-latency tasks such as those that just received an I/O interrupt.

Such a model works well if VM workloads use the CPU resources in a homogeneous fashion. However, CPU residency times of services workloads are very low [10], but at the same time it is important for them to get the CPU as soon as they are ready to run, as otherwise QoS requirements may not be met. The mechanisms provided by the proportional schedulers fail to achieve that, since proportional sharing only promises a higher proportion of CPU without any timing guarantees.

This lack of QoS support in proportional schedulers has been identified as a problem in conventional OS schedulers and is addressed to some extent in [22] and [28], as well as in hypervisor schedulers like Xen [17]. As a part of Themis, we also implement real time scheduling for services similar to QLinux [28], while retaining default proportional scheduling for batch VMs. A new priority state is introduced in the scheduler, referred to as the ‘QoS’ state, which is between the Boost and Under states. The QoS state can be configured through Xen API for any VM which has a QoS requirement. However, as we show below, providing such real time priority is not sufficient by itself to guarantee QoS for the services in consolidated virtualized environments.

In Figure 1 we show results of an experiment where we run RUBiS web server in a higher priority QoS VM state on one CPU socket, and a batch VM running *swim*, a compute intensive weather prediction benchmark from SPEC'00, on another socket of an Intel Xeon quad core machine. Figure shows on x-axis time in seconds, and on dual y-axis CPU utilization of the servers and QoS ratio of RUBiS. The target QoS ratio should be less than 1. We also ran RUBiS by itself and found has QoS ratio close to 0.3. The interference effects due to the batch VM slow down the web server, even though they do not share a physical core. This consequently increases the CPU utilization of the web server VM, creates a bottleneck thus worsening the QoS ratio of the application to unacceptable levels that are 3x higher than the specified limit.

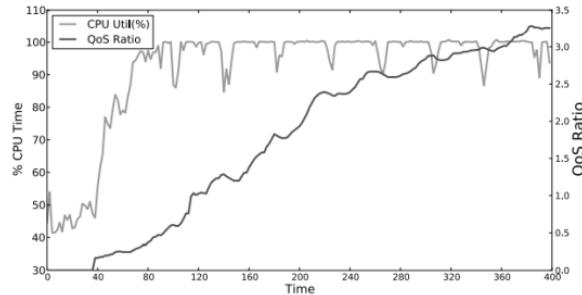


Fig. 1. RUBiS-web with a batch VM running ‘*swim*’

This example shows that just guaranteeing real time priority is not sufficient to ensure QoS for service VMs in consolidated environments. The interference effects due to shared resource usage can dramatically impact the QoS level even when CPU resources are not shared, and must be explicitly accounted for. These interference effects are a function of how the workloads interact with each other which are difficult to model. Consequently, our approach, as described in the next section, is to infer the interference effects through resource utilization and QoS ratio feedback from the service VMs, and to adaptively provision the resource allocation to alleviate these issues.

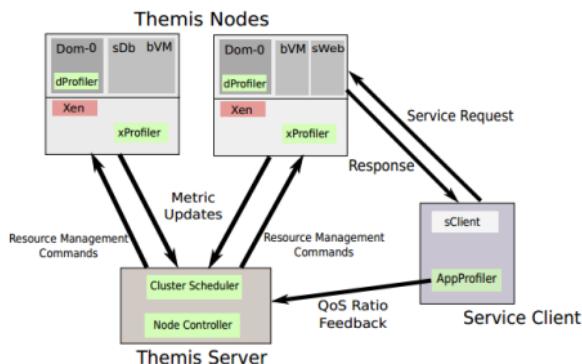


Fig. 2. Themis Design

2.2 Themis Components

The overall objective of Themis is to manage diverse workloads in the data center with the goal of maximizing qMIPS/Watt. It implements a monitoring framework for dynamic VM profiling and policies for dynamic resource management of VMs. Themis has three entities as shown in Figure 2: (1) Themis Nodes: These are the physical machines in the data center that run the actual workloads, which can be a batchVM, serviceVM or both. (2) Services Clients: These are the machine(s) that are running applications requesting service from a serviceVM (which can be a single or multi-tier service) running on the Themis clients. (3) Themis Server: This is the cluster manager, and is responsible for implementing policies for node level resource management and VM scheduling. We now present these entities in greater detail.

1) *Themis Nodes* (tNodes) are physical machines that run workloads. They are equipped with Themis specific profilers to capture live metrics that estimate per-VM resource utilization. There are two such profilers:

xProfiler: It captures throughput (MIPS) and memory access information (MPC) for VMs running on tNodes with CPU performance counters, and communicates that to dProfiler. This data is not used by Themis, but is required for comparison with state of the art scheduling policies [11, 25], and to estimate qMIPS/Watt for all policies.

dProfiler: It compiles per-VM performance and resource utilization information, and communicates it to Themis server. The dProfiler runs inside the Dom-0.

2) *Services Clients* (sClients) are the applications serviced by the serviceVMs running on tNodes. The sClients use appProfiler to dynamically communicate QoS ratio to the Themis server. We assume that it is feasible to implement such appProfilers for all the services whose QoS needs to be monitored. The QoS ratio is dynamically communicated by the appProfiler to the Themis server.

3) *Themis Server* (tServer) does resource management across the cluster with the objective of maximizing the qMIPS/Watt. It registers all tNodes and sClients through dProfiler and appProfilers, periodically collects the metric updates and QoS ratios from them, and sends this to the management policies running on the system. The policies convey their management decisions to the dProfiler, which physically implements them on the intended tNode as illustrated in Figure 2. The tServer uses a cluster scheduler similar to the existing state of the art implementations [14, 29], that consolidates batch and serviceVMs based on the CPU utilization metrics of the individual VMs running across the cluster provided by the dProfiler. However, when we co-locate batchVM and serviceVM on a tNode, the interference effects can impact the QoS ratio of the services. While batchVMs need CPU resources to maximize their throughput, the serviceVMs only need the CPU resources for long enough to service the client requests within the required time frame. This observation motivates the design of a resource management policy, referred to as the ‘Node Controller’ to dynamically control the CPU resource allocation to the serviceVMs.

4) *Node Controller* (tController) exists for every serviceVM which operates independently based on the metric and QoS inputs. At every time interval tController predicts what is the needed number of CPUs (which we refer to as *nref*) for the next interval based on the CPU utilization and QoS ratio. Its objective is to converge to

the number of virtual CPUs sufficient to meet QoS ratio for the serviceVMs, while giving as much CPU headroom to batchVMs as possible. Algorithm 1, used to estimate n_{ref} for the upcoming interval, takes as input the current QoS ratio (QoS_{cur}), CPU utilization ($util_{cur}$) and the vCPU allocation (n_{cur}) for the serviceVM the tController is managing. It further uses two important threshold paramenters: (1) QoS_{th} : This threshold is used by the algorithm to determine whether the QoS of the service being monitored is being safely met. If the QoS_{cur} is less than QoS_{th} , then current resource allocation is more than sufficient to meet QoS metrics. (2) $util_{th}$: CPU utilization threshold that is used to determine if the service needs additional CPU resources. The experiments with service workload and batch jobs showed that service CPU utilization scales linearly with the number of CPUs regardless of the type of batch jobs.

Algorithm 1. Performance Model

Input: QoS_{cur} , $util_{cur}$ and n_{cur}

- 1: **if** $QoS_{cur} < QoS_{th}$ **then**
- 2: $n_{next} \leftarrow n_{cur} - 1$
- 3: $util_{next} \leftarrow (util_{cur} \times n_{cur}) / n_{next}$
- 4: **if** $util_{next} > util_{th}$ **then**
- 5: return n_{cur}
- 6: **end if**
- 7: **else**
- 8: $n_{next} \leftarrow n_{cur} + 1$
- 9: **end if**
- 10: return n_{next}

We model the error in selecting the number of vCPUs for each serviceVM, $\delta n(k)$, at each time step k , as a state which is related to the number of currently assigned vCPUs, $n(k)$, and the target number of vCPUs estimated using Algorithm 1, $n_{ref}(k)$:

$$\delta n(k+1) = \delta n(k) + n(k) - n_{ref}(k) \quad (2)$$

To maximize the qMIPS/Watt, the error for the next step, $\delta n(k+1)$, has to converge to zero, as this ensures that we give minimum resources the serviceVM needs to meet its QoS, hence maximizing the achievable MIPS for the batchVM. We use closed loop feedback to achieve this as shown below:

$$n_i(k+1) = -G_i \delta n(k) + n_{ref}(k) \quad (3)$$

where G_i is the state feedback gain for the i^{th} serviceVM and $n_i(k+1)$ is the number of vCPUs that serviceVM needs to meet its desired QoS ratio. We pick value of G_i that is between zero and one as this guarantees convergence of the controller per results available from control theory. At each control decision point, the controller calculates $n_{ref}(k)$, estimates current cumulative error, $\delta n(k)$, and based on control shown in Equation (3), it estimates the next step's vCPU allocation, $n_i(k+1)$. In practice we found that this takes only a handful of iterations, with QoS_{th} of 0.6 and T_s of 2s as representative parameters. These parameters need to be choosen appropriately for other deployments. Techniques such as ARMA estimators and maximum likelihood tests can be used to do parameter prediction and selection online.

3 Results

We conduct experiments on a testbed of four dual quad-core Nehalem machines with 24GB memory. Two are tNodes, which run the service and batch VMs, 3rd is sClient which generates workload for services, and the last is tServer, doing scheduling and resource management of the VMs. We compare Themis (labeled as *Controller*) to existing state of the art systems:

(1) *Baseline*: This policy runs service VMs and batch VMs on separate tNodes to avoid any interference effects.

(2) *Consolidation*: This policy consolidates service VMs and batch VMs on the basis of CPU utilization [14, 29] but does not perform resource management with the Node Controller.

(3) *Ideal-tChar*: Systems proposed in [10, 20] identify the memory intensiveness of the batch VMs, and distribute VMs across physical machines to avoid MIPS degradation. We ensure that they only consolidate non-memory intensive batch VMs with the service VMs. If the batch VM is memory intensive, then the VMs run independently on separate tNodes. This helps meet QoS ratio by limiting the degree of consolidation. We guarantee offline that consolidation occurs only if QoS ratio is satisfied, thus labeling the policy Ideal.

(4) *Ideal-tCap*: Systems proposed in [21, 26] manage QoS on consolidated machine by placing a CPU cap on the lower priority VM to ensure that the higher priority VM meets its QoS requirements. A CPU cap limits the amount of time the lower priority VM runs, hence reducing the interference effects. We place a CPU cap on the batch VM based on the QoS ratio feedback of the application being serviced by the service VM to ensure that the QoS requirements are met. This policy is labeled Ideal since we determine the minimum cap for batch VMs so that the QoS ratio is satisfied offline.

In all our experiments, initially one tNode runs the service VMs and the other tNode runs the batch VM. We then monitor the MIPS of the batch VM, QoS ratio for the service VMs and the power consumption of the active tNodes every two seconds (Ts) for the whole duration of the run of the sClient. The power consumption is recorded from the power sensors on the machines, which are accessible through an Integrated Power Management Interface (IPMI) [16]. Using these values, we estimate the qMIPS/Watt. The initial configuration of the batch and service VMs is identical for all policies. All the VMs run Linux as the guest OS, and are configured with 2 vCPUs and 4GB of memory. The service VMs are assigned the ‘QoS priority state’ for all the policies to ensure timely access to the CPU. RUBiS is configured so that service VMs comfortably meet QoS ratio when running alone. For the batch VMs, we always have as many threads of the benchmark as the number of vCPUs to represent a fully utilized VM. The mix of our batch jobs has an equal number of CPU and memory intensive benchmarks. In Figures 4 and 5 we list results for CPU bound benchmarks on the left, and memory bound on the right, with average over all on the far right.

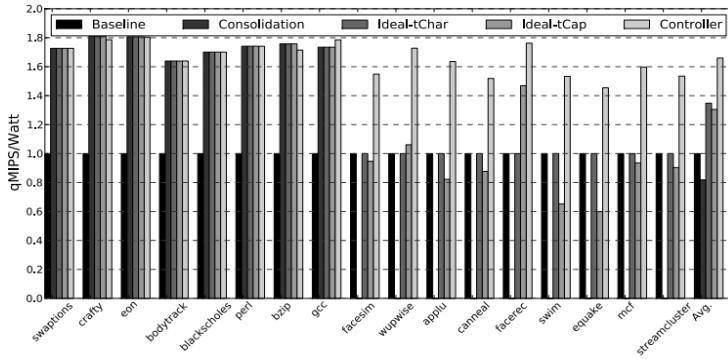
**Fig. 3.** Energy efficiency comparison

Figure 4 illustrates the overall results in qMIPS/Watt for all the policies, normalized against the baseline policy, with workloads listed on y-axis running inside the batch VM. Higher values of qMIPS/Watt correspond to better system energy efficiency. The more memory intensive the batch job, the more it impacts the execution of service jobs. Baseline policy gives the best MIPS for the batch jobs but is inherently energy inefficient, since it keeps two machines active, resulting in the highest active power consumption. Consolidation policy saves as much as 40% of power through VM consolidation. However, it results in poor qMIPS/Watt because the consolidation with memory intensive batch VMs results in bottlenecks for the service VMs, and consequently violations of their QoS (failure to meet the QoS requirements corresponds to zero qMIPS/WATT). The Ideal-tChar policy combines the best parts of two previous policies. This policy, based on oracle knowledge, consolidates the VMs when their QoS is maintained and keeps them separate when not. Ideal-tChar gets a 40% increase in qMIPS/WATT over the baseline for RUBiS. The policy however, misses out on the opportunity to save energy through consolidation for more memory intensive batch VM workloads. The Ideal-tCap policy, on the other hand, accomplishes consolidation under all circumstances as it places a limit on the CPU utilization of the batch VM so that the service VM just meets its QoS requirement. A cap on CPU allocation results in smaller time spent by batch VM on the CPU, which reduces the interference effects. However, Ideal-tCap not only fails to improve the useful work done per joule but actually results in its slight reduction. This is explained in Figure 5, where we observe that the MIPS of the batch VM because of capping drops considerably. This results in the Ideal-tCap policy performing even worse than the Baseline policy in terms of qMIPS/Watt for some very memory intensive batch VMs. Our Controller policy outperforms all the other policies for both service workloads. It is on average 70% better than the Baseline in qMIPS/WATT and 35% better than the Ideal policies. The large gains in qMIPS/Watt of the Controller policy over the Ideal policies are a consequence of the fact that the controller is better able to exploit the heterogeneity in the way resources are used.

Figure 5 demonstrates that the raw MIPS achieved by the Controller is on an average within 7% of the maximum possible, i.e. the Baseline, and in the worst case within 17%. At the same time it is able to reduce the system power consumption by 50% relative to the Baseline policy. In contrast, Ideal-tCap policy is on an average 30% below and up to 70% worse than the Baseline.

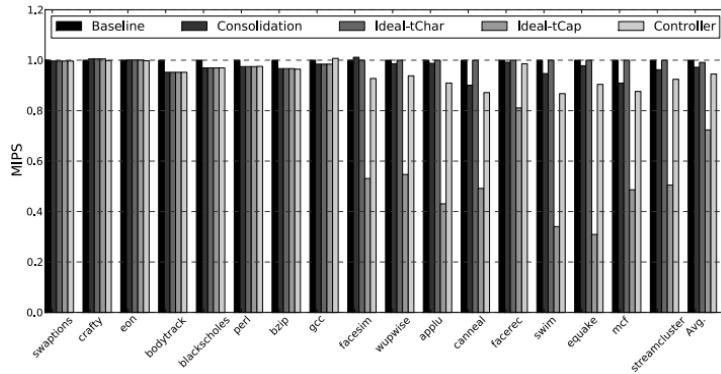


Fig. 4. Batch VM MIPS running with Rubis

4 Summary

This paper explores the challenges of managing latency sensitive services and throughput oriented batch jobs in data centers. We design a new metric, qMIPS/Watt, to capture the amount of work done per joule while maintaining a prespecified level of QoS. Our Themis controller, which leverages the heterogeneity of the workloads when managing resources, outperforms ideal versions of state-of-the art policies in work done per Joule by 35% on average, and by 70% relative to the baseline in today’s data centers. Going forward we plan to include other resources to manage, such as I/O, and we plan to integrate Themis as a part of one of the large scale cloud management systems, such as OpenStack [31], which will enable us to more easily evaluate Themis’ benefits at larger scale.

Acknowledgments. This work was funded in part by NSF grants No. EEC-0812072, No. CNS-0821155, No. OCI-0962997, MuSyc center, Microsoft and Google.

References

- [1] Amza, C., Cecchet, E., Chanda, A., Cox, A.L., Elnikety, S., Gil, R., Marguerite, J., Rajamani, K., Zwaenepoel, W.: Specification & implementation of dynamic web site benchmarks. In: IEEE WWW (2002)
- [2] Apache, <http://incubator.apache.org/olio/>
- [3] 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)
- [4] Barroso, L., Holzle, U.: The Case for Energy-Proportional Computing. IEEE Computer 40(12) (December 2007)

- [5] Blagodurov, S., Zhuravlev, S.: Contention-Aware Scheduling. *ACM Trans. on Computing Systems* (2010)
- [6] Bobroff, N., Kochut, A., Beaty, K.: Dynamic Placement of Virtual Machines for Managing SLA Violations. *IEEE Integrated Network Management* (2007)
- [7] Chase, J., Anderson, D., Thaka, P., Vahdat, A., Doyle, R.: Managing Energy and Server Resources in Hosting Centers. In: *SOSP 2001* (2001)
- [8] Clark, C., Fraser, K., Hand, S., Hansen, J., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live Migration of Virtual Machines. In: *NSDI* (2005)
- [9] Dhiman, G., Kontorinis, V., Tullsen, D., Rosing, T., Saxe, E., Chew, J.: Dynamic Workload Characterization for Power Efficient Scheduling on CMP Systems. In: *ISLPED* (2010)
- [10] Dhiman, G., Marchetti, G., Rosing, T.: vGreen: A System for Energy Efficient Computing in Virtualized Environments. In: *ISLPED* (2009)
- [11] Dhiman, G., Pusukuri, K., Rosing, T.: Analysis of DVFS for Energy Management. In: *USENIX-HotPower* (2008)
- [12] Fan, X., Weber, W., Barroso, L.: Power Provisioning for a Warehouse-sized Computer. In: *ISCA* (2007)
- [13] Ge, R., Feng, X., Feng, W., Cameron, K.: CPU MISER. In: *ICPP* (2007)
- [14] Hermenier, F., Lorca, X., Menaud, J., Muller, G., Lawall, J.: Entropy: a Consolidation Manager. In: *VEE* (2009)
- [15] Hoelzle, U., Barroso, L.: The Datacenter as a Computer (2010)
- [16] IPMI, v2.0 Specification, Intel (2004)
- [17] Lee, M., Krishnakumar, A., Krishnan, P., Singh, N., Yajnik, S.: Supporting real-time in the Xen hypervisor. In: *VEE 2010* (2010)
- [18] Mcnett, M., Gupta, D., Vahdat, A., Voelker, G.: Usher. In: *LISA 2007* (2007)
- [19] Meisner, D., Gold, B., Wenisch, T.: PowerNap: Eliminating Server Idle Power. In: *ASPLOS* (2009)
- [20] Merkel, A., Stoess, J., Bellosa, F.: Resource-Conscious Scheduling for Energy Efficiency. In: *EuroSys 2010* (2010)
- [21] Nathuji, R., Kansal, A., Ghaffarkhah, A.: Q-Clouds: Managing Interference for QoS-Awareness. In: *EuroSys* (2010)
- [22] Nieh, J., Lam, M.: A Smart Scheduler for Multimedia Applications. *ACM Trans. Comput. Syst.* 21 (2003)
- [23] Nurmi, D., Wolski, R., Grzegorczyk, C., Soman, S., Youseff, L., Zagorodnov, D.: Euca-lyptus. In: *ISCCG 2009* (2009)
- [24] Ongaro, D., Cox, A., Rixner, S.: Scheduling I/O in Virtual Machine Monitors. In: *VEE* (2008)
- [25] OpenNebula, <http://www.opennebula.org/>
- [26] Padala, P., Hou, K., Shin, K., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A.: Automated Control of Multiple Virtualized Resources. In: *EuroSys 2009* (2009)
- [27] Rajamani, K., Lefurgy, C.: On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In: *ISPASS* (2003)
- [28] Sundaram, V., Chandra, A., Goyal, P., Shenoy, P., Sahni, J., Vin, H.: Application Performance in the QLinux Multimedia Operating System. In: *MULTIMEDIA 2000* (2000)
- [29] Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and Gray-box Strategies for Virtual Machine Migration. In: *NSDI* (2007)
- [30] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: *OSDI* (2004)
- [31] OpenStack (2012), <http://docs.openstack.org>

Runtime Virtual Machine Recontextualization for Clouds

Django Armstrong¹, Daniel Espling², Johan Tordsson², Karim Djemame¹,
and Erik Elmroth²

¹ University of Leeds, United Kingdom

² Umeå University, Sweden

Abstract. We introduce and define the concept of recontextualization for cloud applications by extending contextualization, i.e. the dynamic configuration of virtual machines (VM) upon initialization, with autonomous updates during runtime. Recontextualization allows VM images and instances to be dynamically re-configured without restarts or downtime, and the concept is applicable to all aspects of configuring a VM from virtual hardware to multi-tier software stacks. Moreover, we propose a runtime cloud recontextualization mechanism based on virtual device management that enables recontextualization without the need to customize the guest VM. We illustrate our concept and validate our mechanism via a use case demonstration: the reconfiguration of a cross-cloud migratable monitoring service in a dynamic cloud environment. We discuss the details of the interoperable recontextualization mechanism, its architecture and demonstrate a proof of concept implementation. A performance evaluation illustrates the feasibility of the approach and shows that the recontextualization mechanism performs adequately with an overhead of 18% of the total migration time.

1 Introduction

Infrastructure as a Service (IaaS) clouds are commonly based on virtualized hardware platforms executing and orchestrating self-contained virtual machines (VMs), which are comprised of multiple virtual devices. A cloud application is typically subdivided into individual components, each component bundled into a specific type of VM. Several VM instances can be started using the same type of VM (using the same master disk image) and each new VM instance is uniquely configured, *contextualized*, with instance specific settings at the early stages of execution. The capacity of the cloud application can be adjusted by changing the amount of VM instances. For clarity, our definition of contextualization is as follows:

Definition. *Contextualization* is the autonomous configuration of individual components of an application and supporting software stack during deployment to a specific environment.

In this work we introduce the concept of *recontextualization*. Recontextualization can be used to adapt to any system changes, including making newly migrated VMs operate properly in the (potentially different) system environment of a new host. We define recontextualization as follows:

Definition. *Recontextualization* is the autonomous updating of configuration for individual components of an application and supporting software stack during runtime for the purpose of adapting to a new environment.

The life-cycle of a cloud application is comprised of three individual phases as shown in Figure 1. The Construction phase refers to the development of a cloud application making use of platform services and dividing that application into a set of VM images. In the Deployment phase a constructed application is deployed on to suitable infrastructure and finally in the Operation phase the cloud application is executed. The application can be configured in the Construction phase and contextualized with specifics of a provider's environment in the Deployment phase. Recontextualization offers dynamic reconfiguration in the Operation phase.

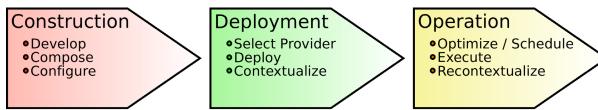


Fig. 1. The life-cycle of a cloud application

Recent work on IaaS systems have a lot in common with the vision of autonomic computing, as outlined by Kephart and Chess [9]. One of the major aspects of autonomic computing that has yet to be realized is self-configuration, the automated configuration and adjustment of systems and components. Our earlier work on contextualization [1] presents a mechanism for boot-time self-configuration of VMs. This work extends state of the art and our earlier efforts by introducing runtime recontextualization, enabling adaptation of VM behavior in response to internal changes in the application to which the VM belongs or to external changes affecting the execution environment of the VM. The concept can enable applications at the PaaS to adapt to different provider application middleware services through the dynamic binding of APIs, enabling the execution of site specific code. This, however, is out of scope in this paper.

The contributions of this paper are: i) The concept and definition of recontextualization ii) The development of an architecture and mechanism for the purpose of recontextualization. iii) A demonstration and evaluation of a recontextualization system. The rest of this paper is organized as follows: Section 2 outlines the problem to be solved, a set of requirements for any approach to recontextualization and an illustrative scenario of recontextualization for application monitoring. Section 3 discusses different approaches considered for runtime

recontextualization. Section 4 presents our proposed solution for recontextualization of VMs including an evaluation of the approach. Finally, a conclusion and future work are presented in Section 5.

2 Problem Statement and Requirements

A motivational factor behind the need for runtime recontextualization stems from VM migration in clouds [3,15]. Using migration, a VM can be transferred from one physical host to another without explicitly shutting down and subsequently restarting the VM [4]. The entire state of the VM, including e.g., memory pages, are transferred to the new host and the VM can resume its execution from its state prior to migration. As a consequence of this, no contextualization is triggered again when the VM is resumed, as the level of abstraction provided by virtualization is insufficient for platform services. In this paper we consider migration from and to identical hypervisor technology, interoperable migration is out of scope but is considered in [12]. As presented in [6], there are several different cloud scenarios:

- Bursting - The partial or full migration of an application to a third party IaaS provider, this may occur when local resources are near exhaustion.
- Federation - The migration of an applications workload between a group of IaaS providers, e.g., when a single provider's resources are insufficient for maintaining the high availability of an application through redundancy.
- Brokering - The migration of an application's VMs, e.g., for the purpose of maintaining an agreed Quality of Service (QoS) in the case of an end-user utilizing a broker to select a provider given a set of selection criteria.

In all these cloud scenarios VM migration is a necessity, e.g., for the purpose of consolidating resources and maintaining levels of QoS. We have used these scenarios to guide us when the defining of requirements for any potential recontextualization mechanism. We consider the following requirements as imperative:

- i. A triggering mechanism for recontextualization on VM migration.
- ii. A secure process to gather and recreate contextualization data after migration.
- iii. A hypervisor agnostic solution that maintains IaaS provider interoperability.
- iv. An approach that is non-pervasive and minimizes modifications at the IaaS level.

We make a case for each of these scenarios requiring recontextualization at runtime. In the Bursting scenario, if an IaaS provider is not obligated to divulge third party providers used for outsourcing of computational resources, an application may end up deployed on to a third party's infrastructure that requires use of their local infrastructure services. A dynamic federation of IaaS providers created during negotiation time that alters during the operation phase requires infrastructure services to be discovered dynamically. The same is applicable in

the case of a Broker, knowledge of a providers local infrastructure services is not available during deployment until after the Broker has selected a provider.

The lack of knowledge on the attributes of an IaaS provider's local infrastructure service available during deployment time further motivates our work. An example of such a service that exhibits configuration issues after resource migration is application-level monitoring.

In this example the monitoring service endpoint, to which application Key Performance Indicators (KPI) are reported, can be configured by contextualization during the deployment phase of an application's life cycle. However, the endpoint may change during the application lifetime, either as a result of changes in the local system environment or due to migration of the application to a new host. This example motivates the need for a mechanism to fetch configuration data during application operation and provide new context to application dependencies, thus *recontextualization*. In the following section, we illustrate recontextualization with service-level monitoring [7] as an example scenario.

2.1 Example Scenario

A typical cloud application must be continually monitored during runtime, an example of this is shown in Figure 2. Monitoring data can be used for several purposes, e.g., for automatic application scaling or to assess the likelihood and prevent the breaching of a Service Level Agreement (SLA). Application level metrics, known as KPIs, are sent from inside the VM to an external monitoring endpoint for processing.

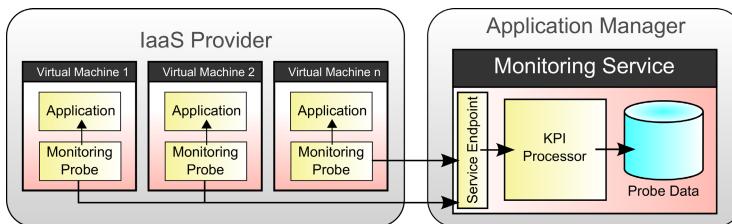


Fig. 2. Monitoring applications in a IaaS provider

Each monitoring probe that gathers KPI data must be configured with the endpoint or location of the monitoring service. The endpoint can be associated with a IaaS specific service or a service running at a remote location and depends on what entity within connected clouds has control over application management. When deploying to a IaaS provider, the endpoint for the monitoring service is configured using contextualization in the Deployment phase. However, in a multi-site scenario the VM maybe migrated to an unknown provider during runtime and must therefore be dynamically recontextualized with a new endpoint in the Operation phase.

3 Recontextualization Approaches

As far as we are aware no previous research has considered an approach for recontextualization. Keahey and Freeman [8] present fundamental work on contextualization in virtual clusters and recontextualization is mentioned but deemed out of scope for their work. In this section several different approaches for contextualization are considered for use in recontextualization. Any recontextualization approach has two major obstacles that must be dealt with; how is recontextualization triggered and where can the necessary information be found? Below are some approaches for recontextualization, listed and discussed, from the perspective of the above two challenges.

Contextualized direct addressing is based on a known endpoint address that is specified in the initial contextualization phase, as described by Armstrong et al. in [1]. A similar approach is used for Puppet [14], a mass-machine configuration tool for HPC-like environments. During operation, this endpoint address is queried for the updated context information. Furthermore, this approach is interoperable and requires no host and hypervisor modifications, but requires that the end point address is constant when a VM is migrated to other domains. This approach offers no procedure for triggering a new round of recontextualization, and has to rely on periodically polling the endpoint for updates.

Hypervisor network proxying also relies on periodically querying an external endpoint address for context information, but in this method a standard virtual network address is used and the hypervisor (and associated virtual network management) is responsible for routing this call to a host specific endpoint. This approach, used by Clayman et al. in [5], is transparent to the VM but requires modifications at the hypervisor level.

Hypervisor interaction by the guest can be used to offer contextualization data straight from the hypervisor itself, using a customized API both to react to changes in context information and to transfer new information. However, this solution requires modifications both to hypervisor and guest operating system software and would require considerable standardization to be widely available, with regards to the compatibility of virtual hardware APIs between hypervisor technologies.

Dynamic virtual device mounting is based on dynamically mounting virtual media containing newly generated content in a running VM via the reuse of existing hypervisor interfaces and procedures [1]. Interoperability is achieved by reusing existing drivers for removable media such as USB disks or CD-ROM drives. Recontextualization can be detected by the guest OS by reacting to events triggered when new USB or CD-ROM media is available.

We propose that the dynamic virtual device mounting approach is the most promising solution to recontextualization due to inherent interoperability and support in all major operating systems. The ability to manage virtual devices is also offered by the Libvirt API [11], inferring that there is fundamental support

for these operations in most major hypervisors. The following section describes our recontextualization solution in more detail.

4 A Recontextualization Solution

In this section, an implementation of a system for runtime recontextualization is described, followed by an evaluation to validate the suggested approach. The previously discussed virtual device mounting technique is used in response to migration events and thus enables automatic self-configuration of newly migrated VMs. The following subsections discuss the mechanism, architecture, and evaluation in more detail.

4.1 Mechanism

Figure 3 illustrates the recontextualization approach used in the implementation. Each VM is assigned a virtual CD-ROM device for contextualization on which the host-specific and thus provider contextualization data can be found. When a VM is migrated from one host to another events describing this action are triggered by the hypervisor, which can be registered to via the Libvirt API. In response to these events, the recontextualizer software triggers a detachment of the virtual device mounted with contextualization information, and once the migration is completed a new virtual device with context information relevant for the new host is automatically attached to the VM as it resumes operation after migration.

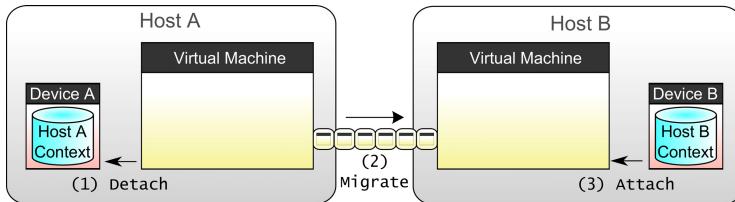


Fig. 3. Recontextualization approach overview

Event support including migrations is present in several hypervisors, including Xen [2] and KVM [10]. The Libvirt API enables a unified approach to VM management available and includes event support. An initial version of the recontextualization system was implemented using KVM with QEMU [13] specific event and control APIs and the second version was implemented using Libvirt to make the solution hypervisor independent. Libvirt provides a number of event types that can be monitored via a callback: i) Started, ii) Suspended, iii) Resumed, iv) Stopped, and v) Shutdown. Upon receiving an event callback details are returned on the specific cause of the event, for example the shutting down of a VM on a host machine triggered by migration terminating successfully.

4.2 Architecture

The architecture of the implemented system is shown in Figure 4. Up-to-date context data is dynamically bundled as ISO images on the host. The recontextualizer, implemented in Python, manages the attachment and detachment of virtual CD-ROM devices inside a VM that contain the data held within the ISO image media in response to events from the hypervisor. The Python Libvirt API bindings were used to access the Libvиртd daemon for the purpose of abstracting the specifics of the underlying hypervisor and to improve interoperability.

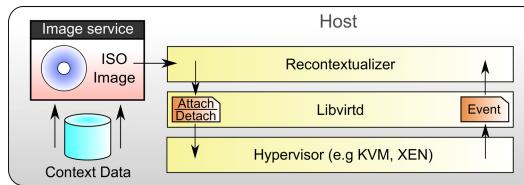


Fig. 4. Architecture overview

4.3 Evaluation

A series of tests to evaluate the feasibility of the approach have been performed. For all tests, Libvirt version 0.9.9 was used to monitor and manage the VMs. QEMU-KVM version 1.0.50 and Xen version 4.0.0 were used as hypervisors, both running on the same hardware using CentOS 5.5 (final) with kernel version 2.6.32.24. The hosts used in these tests are on the same subnet, have shared storage and are comprised of a quad core Intel Xeon X3430 CPU @ 2.40GHz, 4GB DDR3 @ 1333MHz, 1GBit NIC and a 250GB 7200RPM WD RE3 HDD.

The results of the evaluation are shown in Figure 5. The first set of bars illustrate the time to migrate a VM from one host to another with recontextualization running and context data attached, and the second set of columns illustrate the same migrations with recontextualization turned off and no virtual devices mounted. The third column illustrates the time spent within the recontextualizer software during the tests from the first column, measured from when the event for migration was received in the recontextualizer until the device had been removed and reattached. The values shown are the averages from ten runs, and all columns have error bars with the (marginal) standard deviations which are all in the 0.03 to 0.07s range.

Based on the evaluation we conclude that the recontextualization process adds about an 18% overhead using either hypervisor compared to doing normal migrations. For KVM, most of the extra time required for recontextualization is spent outside the bounds of our component, likely associated with processing events and extra overhead imposed by preparing migration with virtual devices attached. In the case of Xen the device management functionality in Libvirt proved unreliable and we therefore had to bypass the Libvirt API and rely on sub-process calls from the recontextualizer to Xen using the *xm* utility. This workaround increased the time needed for recontextualization in the Xen case.

There are four major phases associated with the recontextualization process. First, information about the VM corresponding to the event is resolved using Libvirt when the migration event is received. In the second phase, any current virtual contextualization device is identified and detached. Third, new contextualization information is prepared and bundled into a virtual device (ISO9660) image. Finally, the new virtual device is attached to the VM. A detailed breakdown of the time spent in different phases of recontextualization is presented in Figure 6. The above mentioned workaround for Xen interactions affects the second and fourth phase (detaching and attaching of devices), most likely increasing the time required for processing. In the first and third phases Xen requires significantly longer time than KVM despite the VMs being managed using the same calls in the Libvirt API, indicating performance flaws either in the link between Libvirt and Xen or in the core of Xen itself.

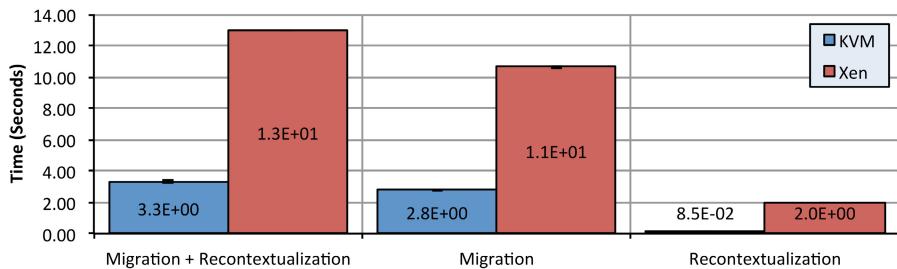


Fig. 5. Time measurements of recontextualization.

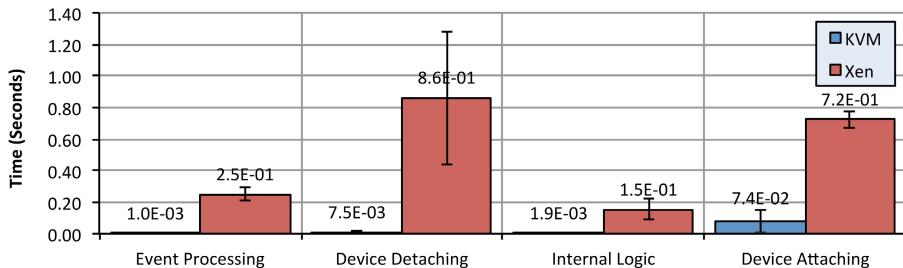


Fig. 6. Breakdown of time spent during recontextualization.

4.4 Practical Experiences

When creating the system a vast number of bugs and shortcomings both with Libvirt and the underlying hypervisors were experienced. It turned out that migrating VMs using KVM with USB devices attached periodically caused the migration to fail without any indicative error of the root cause. It was discovered after looking through the source code of qemu-kvm (necessary due to a lack of

documentation) that support for migration with USB devices is still to be fully implemented. To overcome this issue the use of a virtual CD-ROM device as a replacement was explored. Unfortunately this approach has the drawback of needing the guest OS to be configured to automatically re-mount the ISO image. We used *autofs* within our Debian guest VM for the purpose of testing. For other operating systems such as Windows that natively support the automatic mounting of CD-ROMs this would not be a problem. Using this device type in our system worked with KVM but Xen would not reliably release media mounted within a VM, causing the recontextualizer to fail in its attempt to provide new context data. To combat this issue we forced the removal of the entire CD-ROM device, reattaching another with a different ISO image.

Considering Libvirt's support of Xen events and the detaching of devices we initially tried to use a Hardware-assisted Virtualization (HVM) guest but found that Libvirt would not propagate any VM events from the hypervisor through its API. After discovering this issue we tried using a Paravirtualised (PV) Xen guest but found that only start and stop events were available. This has had the negative effect of altering the logic of the recontextualizer, where by detaching and attaching devices incurred an additional unnecessary overhead when a virtual machine starts, while for KVM this overhead only occurs after migration.

5 Conclusion and Future Work

We have described and defined recontextualization: the autonomous updating of configuration during runtime. Moreover, we have shown that recontextualization is a key enabler to using multiple cloud sites concurrently. We have evaluated different alternatives for recontextualization based on a set of requirements. Our approach, based on automatic mounting of dynamically generated images as virtual devices, is highly interoperable supporting a variety of hypervisors and virtually all operating systems. Apart from CD-ROM mounting routines, which are standard in most operating systems, no custom software is required inside the guest VM to make the contextualization data available.

Future work includes creating a unified mechanism for contextualization and recontextualization and integrating the solution with major software projects. In addition, recontextualization mechanisms for the dynamic binding of PaaS APIs will be explored. Finally, further studies and improvements on the implemented approach will be evaluated to reduce the overhead imposed by recontextualization.

Acknowledgments. The research that led to these results is partially supported by the European Commission's Seventh Framework Programme (FP7/2001-2013) under grant agreement no. 257115 (OPTIMIS). We would also like to thank Tomas Forsman for technical assistance and expertise.

References

1. Armstrong, D., Djemame, K., Nair, S., Tordsson, J., Ziegler, W.: Towards a contextualization solution for cloud platform services. In: 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), pp. 328–331. IEEE (2011)
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. SIGOPS Oper. Syst. Rev. 37(5), 164–177 (2003)
3. Bradford, R., Kotovinos, E., Feldmann, A., Schiöberg, H.: Live wide-area migration of virtual machines including local persistent state. In: Proceedings of the 3rd International Conference on Virtual Execution Environments, pp. 169–179. ACM (June 2007)
4. Clark, C., Fraser, K., Hand, S., Hansen, J., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, vol. 2, pp. 273–286. USENIX Association (May 2005)
5. Clayman, S., Galis, A., Chapman, C., Toffetti, G., Rodero Merino, L., Vaquero, L., Nagin, K., Rochwerger, B.: Monitoring Service Clouds in the Future Internet. In: Towards the Future Internet - Emerging Trends from European Research, pp. 115–126. IOS Press, Amsterdam (2010)
6. Ferrer, A., Hernández, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R., Djemame, K., Ziegler, W., Dimitrakos, T., Nair, S., Kousiouris, G., Konstanteli, K., Varvarigou, T., Hudzia, B., Kipp, A., Wesner, S., Corrales, M., Forgó, N., Sharif, T., Sheridan, C.: OPTIMIS: a holistic approach to cloud service provisioning. Future Generation Computer Systems (2011)
7. Katsaros, G., Gallizo, G., Kübert, R., Wang, T., Oriol Fito, J., Henriksson, D.: A Multi-level Architecture for Collecting and Managing Monitoring Information in Cloud Environments. In: CLOSER 2011: International Conference on Cloud Computing and Services Science (CLOSER), Noordwijkerhout, The Netherlands (May 2011)
8. Keahey, K., Freeman, T.: Contextualization: Providing One-Click Virtual Clusters. In: Proceedings of the 4th IEEE International Conference on eScience (ESCIENCE 2008), pp. 301–308. IEEE, Washington, DC (2008)
9. Kephart, J., Chess, D.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
10. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux virtual machine monitor. In: Proceedings of the Linux Symposium, vol. 1, pp. 225–230 (2007)
11. Libvirt development team. Libvirt: The virtualization API (February 2012), <http://libvirt.org/>
12. Liu, P., Yang, Z., Song, X., Zhou, Y., Chen, H., Zang, B.: Heterogeneous live migration of virtual machines. In: International Workshop on Virtualization Technology, IWVT 2008 (2008)
13. QEMU development team. QEMU - An open source machine emulator and virtualizer (February 2012), <http://www.qemu.org>
14. Turnbull, J.: Pulling strings with puppet: configuration management made easy. Springer (2008)
15. Wood, T., Ramakrishnan, K.K., Shenoy, P., van der Merwe, J.: CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. In: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 121–132. ACM (2011)

GaaS: Customized Grids in the Clouds

G.B. Barone¹, R. Bifulco¹, V. Boccia², D. Bottalico¹, R. Canonico¹,
and L. Carracciulo³

¹ Università degli Studi di Napoli Federico II

{gbbarone,roberto.bifulco2,davide.bottalico,roberto.canonico}@unina.it

² Italian National Institute of Nuclear Physics, Italy

vania.boccia@na.infn.it

³ Italian National Research Council, Italy

luisa.carracciulo@cnr.it

Abstract. Cloud Computing has been widely adopted as a new paradigm for providing resources because of the advantages it brings to both users and providers. Even if it was firstly targeted at enterprises wishing to reduce their equipment management costs, it has been rapidly recognized as both an enabler for new applications and as a mean to allow enterprises of all sizes at running high demanding applications. Recently, Cloud Providers are trying to attract new applications, such as scientific ones, that today already benefit from distributed environment like Grids. This work presents a way to remove the paradigm mismatch between Cloud and Grid Computing, enabling the use of Cloud-provided resources with well-established Grid-like interfaces, avoiding the need for users to learn new resources access and use models. The proposed approach is validated through the development of a prototype implementation and its integration in a working Grid environment.

Keywords: Virtualization, Cloud Computing, Grid Computing.

1 Introduction

On-demand computing is a model in which computing resources are made available to users as needed. It could be considered a valid solution for people who need a huge amount of resources, to reduce the *Total Time to Solution*, and cannot bear the costs of systems. In particular, these costs grow up when the needed resources are provided by specialized systems, e.g., HPC ones.

The scientific community developed the Grid Computing paradigm to enable the sharing of huge amount of resources through a well-defined distributed infrastructure model, in order to solve large scale problems in a collaborative manner. The Grid Computing resources aggregation model is rather “static”: a group of organizations set up several Grid management services and computing resources in a layered structure that separates the management responsibilities (and corresponding management services) among the organizations involved in the Grid.

Users belonging to the organizations forming the Grid can retrieve information on resources (e.g., their number, status, configuration, etc.) and access them, but can neither change the topology of the grid (e.g., by increasing the number of resources) or manage resources configuration and composition. It would be desirable to have a more “*elastic*” infrastructure in which users can ask for resources on-demand, to suit their needs in terms of resources type and configuration (i.e compilers, scientific libraries, problem solving environments, etc.).

With the advent of new applications and the pervasiveness of IT into everyday activities, also the industrial and private sectors have developed a need for fast access to high demanding IT infrastructures at low costs. The industry answer to these needs has been the Cloud Computing model.

According to the official NIST definition, “*Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources ... that can be rapidly provisioned and released with minimal management effort or service provider interaction*” [10].

Cloud Computing assumes different names depending on the provided resources. When provided resources are computing nodes and storage, Cloud Computing is called Infrastructure as a Service (IaaS). Other resources types include application developing environments (Platform as a Service, PaaS in short) and specific software applications (Software as a Service).

Given the flexibility in resources management through the Cloud Computing paradigm, it seems a promising approach to provide flexible Grid Computing infrastructures through the combination of the Grid and Cloud paradigms.

Some interest has been already shown in this direction [11]. So far, the proposed approaches may be labeled as either “Grid over Cloud” or “Cloud over Grid”, since the composition of the two paradigms may be performed through either the exploitation of IaaS-provided resources to build Grid infrastructures or through the use of Grid-provided computing resources to create IaaS clouds.

In this paper we describe our experience in designing and implementing a solution that creates more flexible Grid infrastructures, by exploiting IaaS-provided resources, in a novel way resembling the PaaS paradigm. We call our solution Grid as a Service (GaaS).

The paper is organized as follows: in the next section is presented the Grid reference architecture, in section 3 we provide an overview on the *state of art* about the integration of Cloud and Grid service models, in section 4 we describe the GaaS model and in section 5 is reported a case study related to the deployment of a prototype on the SCoPE Grid Computing Infrastructure[12]. Finally, in section 6 we present future works and conclude.

2 Grid Computing Reference Architecture

We assume as Grid reference architecture the one implemented by the middleware gLite-EMI, developed in the context of EGI (European Grid Infrastructure). The gLite-EMI middleware provides a Grid infrastructure that is accessible to community members organized into Virtual Organizations (VO).

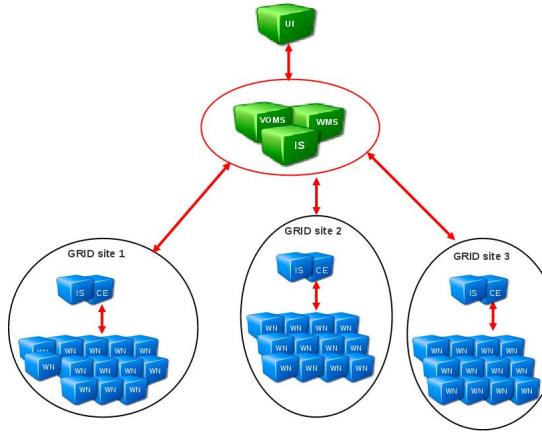


Fig. 1. A gLite-based infrastructure with some central services (UI, VOMS, WMS, IS) and three grid sites (with a CE and several WN)

A VO is defined in [7], as “*a set of individuals and/or institutions defined by such sharing rules...*”. “*VOs vary tremendously in their purpose, scope, size, duration, structure, community and sociology*”. In particular, people from a scientific community, sharing the same “experiment”/applications, can constitute a VO.

VO “managers” make available all the software needed to run the applications of interest of the community on computing resources (the applicative level of the middleware).

The Grid infrastructure is a distributed infrastructure whose management is centralized, while the computation functions are distributed among several sites. The infrastructure provides users with high level services for scheduling and running computational jobs, accessing and moving data, and obtaining information on the infrastructure itself. Services are embedded into a consistent security framework [8]. Those provided are services for authentication/authorization (e.g. VOMS - Virtual Organization Management System), resources allocation and discovery (e.g. LB/WMS - Logging & Bookiping and Workload Management System), infrastructure Information System (IS). Computing resources (WNs - Worker Nodes) are provided by means of CE (Computing Element) that is an endpoint with a set of queues handled by an LRMS (Local Resource Management System). User can access these services from a User Interface (UI).

Management services (UI, VOMS, WMS) are instantiated only once and shared among all the sites, while computing-related services (IS, CE) are replicated in each site. A graphical representation of a minimal gLite-based Grid infrastructure is presented in figure 1.

To use the Grid infrastructure, the user has to (1) authenticate himself on the infrastructure; (2) define a *job* in terms of resources requirements and tasks to be performed; (3) submit the job to the infrastructure by selecting the appropriate resource queue; (4) monitor the job status; (5) retrieve the job execution results.

The *resource queue* is an abstraction provided by the Grid architecture to either group resources based on their properties and to share such resources among several users.

The typical Grid usage model described so far does not allow the user in customizing the grid environment. Users cannot change the Grid infrastructure that runs their experiment, in particular, a user cannot create a new Grid site nor add an existing site for his VO. Also the Grid sites are static and cannot be customized by users, hence, it is not possible to add new worker nodes to a site to extend its capabilities, and it is also not allowed to organize resources into customized queues to shape them in accordance to the computation needs. Finally, even the configuration of the worker nodes cannot be changed.

3 Related Works

In this section we describe related work about the Cloud and Grid integration, presenting examples of “Cloud-over-Grid” and “Grid-over-Cloud” approaches.

An example implementation of the “*Cloud-over-Grid*” is presented in [15], where CLEVER, a cloud management system, is used to provide an IaaS system over Grid. The solution requires the installation of both a specialized CLEVER’s management software and a virtual machine monitor (e.g. VirtualBox) into Grid worker nodes. When the CLEVER cloud requires more resources, new worker nodes can be assigned to it, to dynamically extend the resources available to the cloud.

WNoDeS [14] applies a “*Cloud-over-Grid*” approach as well. WNoDeS (Worker Nodes on Demand Service), developed by the Italian National Institute for Nuclear Physics (INFN), is a solution to virtualize computing resources and to make them available through local, Grid or Cloud interfaces. The Grid infrastructure is exploited through the use of a “*special*” gLite job: the “*power on*”. Users define “Power on” jobs selecting tailored virtual machine images to be launched on computational resources managed by the CE.

In [5] is described an example of “Grid-over-Cloud”, that transparently provides dynamically-instantiated VM-based worker nodes, in an EGEE production grid.

StratusLab [9] is applying a “*Grid-over-Cloud*” approach as well. The StratusLab project aims at developing a complete, open-source cloud distribution that can be deployed in production in both academic and industrial environments. StratusLab provides Grid services using StratusLabs IaaS system resources. The provided Grid infrastructure can exploit the dynamic nature of the cloud, provisioning resources as needed and running user-level (and community level) services using pre-packaged appliances, selected by users and made available by a “*marketplace*”.

In [4] we presented the design and implementation of an on demand computing service, which is able to obtain a right trade-off among management cost reduction, environmental sustainability and user satisfaction. In particular, the work described an experience in designing and implementing a flexible infrastructure, built on the basis of local or remote cloud resources, with the aim of

saving energy to reduce the overall operational cost and to improve environmental sustainability.

Recently, also commercial Cloud Providers are trying to explore the HPC market, by providing resources for, e.g., scientific computations. Most notably, Amazon, is offering “cluster compute” instances, through its Elastic Compute Cloud service, whose resources are tailored for HPC, i.e., they are provided with huge amounts of RAM, processing power, and are deployed on a 10 Gigabit Ethernet network with low delay[1]. To ease the execution of specific workloads, some tools provide automatic configuration of Amazon resources. An example in such sense is CloudFlu[6], that allows the easy execution of OpenFOAM[13] jobs on an automatically configured cluster of Amazon EC2 HPC resources.

4 GaaS: Grid as a Service

In 2012, the European Middleware Initiative (EMI, <http://www.eu-emi.eu>) has published a report in which they describe four possible integration scenarios of virtualized infrastructures in the Grid computing architecture [11]. In this paper, we present *Grid-as-a-Service* (GaaS), a service model designed according to the *Dynamic Grid Services* scenario described in that report:

[Dynamic Grid Services] utilizes the cloud infrastructure to provision grid services using IaaS/PaaS/SaaS models. The grid services, or suitable subsets of the current grid services, can therefore be instantiated on demand ... by deploying and configuring the services on base virtual machines according to specific user community requirements and then disposed of when not needed anymore.

The GaaS model combines the advantage of providing users with an usage model that is familiar to the traditional Grid, with the possibility of flexible management of computational resources in a IaaS-like manner. Hence, our model can be classified as a Platform-as-a-Service for extending Grid environments with elastic (e.g., virtual) resources. By using GaaS, “privileged” grid users, e.g. the VO administrator, can define new Grid Sites, add computational resources to existing Grid Sites and modify the resources aggregation scheme, e.g., site queues. In particular, GaaS provides privileged users with the following functions:

1. WNs management (fig. 2.a): definition, addition and deletion of WN to be used by the Grid infrastructure;
2. Queues management (fig. 2.b): dynamic management of resources in queues, and queues policies configuration;
3. Sites management (fig. 2.c): creation and management of new Grid Sites;

GaaS flexibility provides several advantages to traditional Grid infrastructures, e.g., WNs can be customized with software tailored to a given set of users, as well as queues can be configured to fulfill a specific computation needs. Moreover, GaaS support the creation of complete Grid sites in order to, e.g., enable a

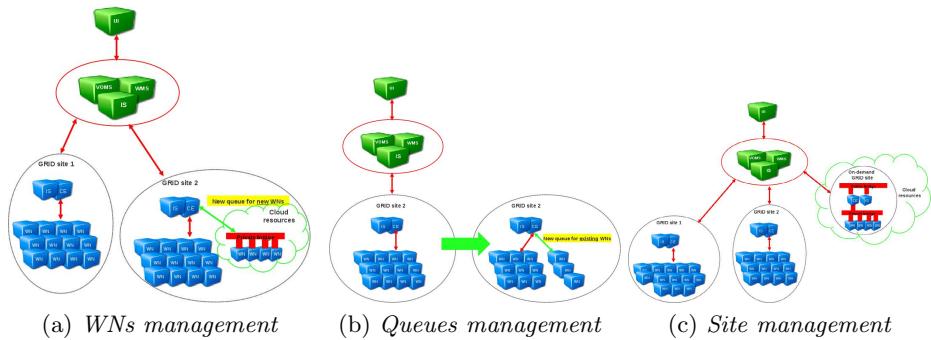


Fig. 2. The GaaS solution

community that has to share resources for the life time of a project, to avoid the burden of configuring from scratch all the required services and resources. Computational resources can be both virtual and physical. The use of virtual resources is not denied to HPC users but, if virtual resources are chosen, users are notified by IS, about the possible performance limitations.

Even if our approach is based on a Grid-over-Cloud model similar to the StratusLab one, in GaaS resources are made available through their configuration in Grid abstractions, e.g., queues. Hence, provided resources can be reconfigured or differently aggregated on the basis of users needs (in a way resembling the PaaS paradigm). Moreover, GaaS enables the provisioning of several high level functions: from queues creation/reconfiguration to the instantiation and configuration of whole grid sites.

5 The SCoPE Case Study

For the validation of the proposed model we implemented a prototype and integrated it into the context of the S.Co.P.E. Datacenter at University of Naples Federico II, a self contained grid infrastructure that offers storage and computational resources and all the high level core services for infrastructure management (VOMS, WMS, IS, etc.). Moreover S.Co.P.E. resources are integrated also into national IGI and international EGI relevant distributed computational infrastructures and used from people belonging to different scientific research fields and to VOs from very relevant international experiments (e.g. LHC, ATLAS, Super-B, etc.). Thus S.Co.P.E. is a suitable context to validate effectively our approach by means of a prototype.

Our prototype is based on the gLite-EMI [8] Grid middleware and on the OpenNebula [2] cloud management system. The modularity of OpenNebula allows for fast introduction of new features to the management system, hence, it allows the easy integration of the Cloud-provided resources into the Grid infrastructure. A subset of the S.Co.P.E. resources are assigned to the OpenNebula managed resources pool. Such resources host an hypervisor, currently Xen [3], to create virtual machines (VM), that are then used as dynamically provided

resources for the Grid infrastructure. VMs are used to both create Grid's WNs and management services such as CE, IS, etc.

The main efforts in the prototype development were (i) the definition of templates for gLite-EMI services configuration, and (ii) the enabling of their fast provisioning.

In many IaaS management system (and in OpenNebula as well), VM templates are usually stored in a “template repository”. A new VM is created copying the selected template to the running location of the VM. The duration of this process is the main factor in the resources deployment time. Since the copy process involves the storage infrastructures that are hosting the *template repository* and the newly created VMs disks, assuming that a minimal VM template is several hundreds of megabytes big, the process, for each VM creation, takes a time in the order of dozens of seconds.

To optimize the infrastructural resources used during the provisioning process, and to reduce the overall provisioning time, we took into account the peculiarities of the GaaS system. In particular, we made the following observations:

1. all the VMs are prescribed to host the same operating system, which is imposed by the gLite-EMI middleware;
2. all the VMs hosting the Grid services (e.g., WN, CE, etc.) can be produced by customizing the configuration of a single VM template;

We designed our VM disk provisioning system in order to provide fast VM creation and avoid as much data copy as possible. Our solution is based on the GNU/Linux's Logical Volume Manager (LVM). LVM allows the creation of logical volumes (LV) and the creation of snapshots starting from a reference LV. Snapshots can be read and written, since their creation is performed through the use of a “*delta meta-data*”, that contains all the differences with the original LV. This approach makes the creation of a snapshot really fast (a few milliseconds) since it involves no copy of data. Once the snapshot is created, following observation 2, a configuration script is executed to customize the virtual resource according to its functional destination. Since both read and write actions involve an a read/update of the *delta meta-data*, the operating performance of the snapshot could be compromised in particular conditions. In our case, we are mainly interested into reading performance, and, moreover, we assume that the majority of reads happen in the bootstrapping process of a VM (e.g., for the loading of the required applications).

In figure 3 is presented a performance comparison of read and write operations on 64 KBs data blocks. These tests were executed on an HP DL380 Proliant server equipped with two Intel Pentium IV Xeon 2.8 GHz CPUs, 5 GB of PC-2100 RAM. The server was running a Debian Linux with the OS kernel configured to use only 1 GB of RAM. We compare the results obtained by operating on both a raw partition (labeled as “normal”) and on a LVM snapshot (labeled as “snapshot”). For various amounts of read/written data (ranging from 512 bytes up to 8 GBytes), we compare the throughput achieved for write (left graphs) and read (right graphs) operations. Figure 3 shows the results of six series of experiments. On the left hand of the figure, graphs a), c) and e) present

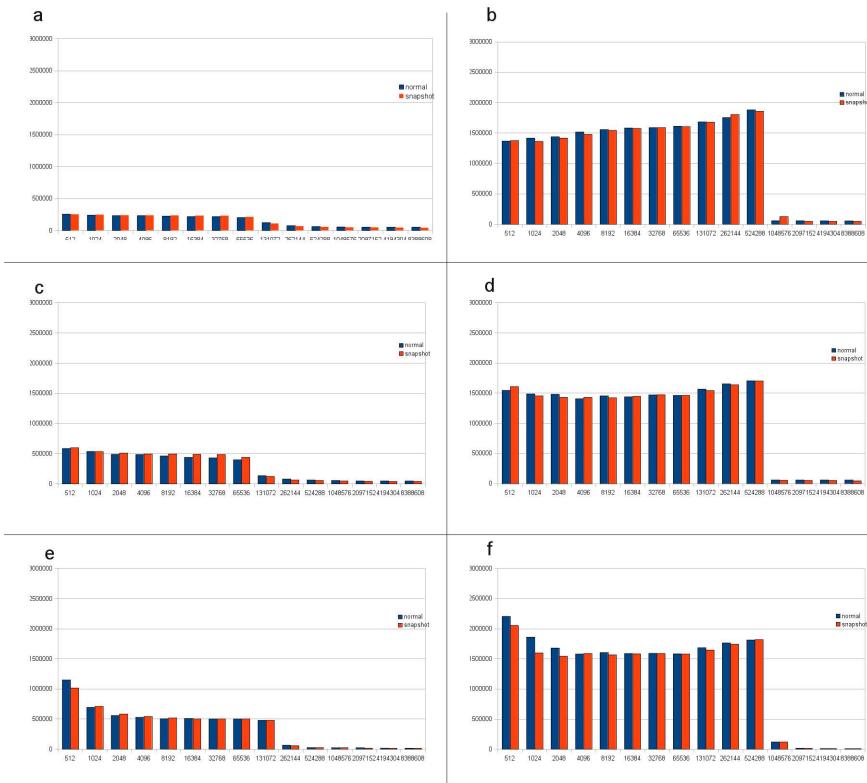


Fig. 3. Storage write (left column) and read (right column) performance. The y axis shows the read and write throughput (bytes/sec), while the x axis shows the amount of data read/written. Graphs a, b refer to sequential operations with no OS caching effects. Graphs c, d refer to sequential repeated operations to maximize OS caching effects. Graphs e, f refer to random operations.

the throughput obtained by write operations performed in the following cases: graph a) refers to operations performed on freshly mounted disks, with no OS caching effects, graph c) refers to sequential write operations performed several times on the same data, in order to maximize the OS caching effects, graph e) refers to random write operations. On the right hand of the figure, graphs b), d) and f) present the results for similar experiments involving read operations.

From the graphs it can be easily observed the effect of caches on performance, when data size is bigger than the available filesystem cache (our system had about 512 MBs of cache space). What is of particular interest for us, is that the performance drop when snapshots are used is marginal, since the *delta meta-data* is likely stored in the system cache anyway, hence, we can fast provision resources to the grid, without paying any sensible penalty on disk performance. Moreover, performance drops are visible when the written/read data size is bigger than the filesystem cache, in particular for write operations. Since snapshots are just used to create VMs' OS bootable disks, i.e., disks that contains the OS and the

applications code but that are not meant to be used as data storage, they are mainly involved in read operations, and the dimension of read data is likely to be smaller or comparable with the dimension of the filesystem cache. Hence, we expect little or no performance drop in the VM operations.

6 Conclusion and Future Work

In this paper we presented GaaS, a PaaS model for Grid Computing systems, that lets VO administrators to dynamically customize the grid environment they are offering to VO's unprivileged members. VO administrators can define new Grid Sites, add computational resources to Grid Sites and modify the resources aggregation scheme (queues). We implemented a prototype of our model and deployed it in a real-world Grid Datacenter. Moreover, our prototype implements a virtual resources fast provisioning scheme, that exploits some properties of the Grid environment. Our implementation uses standard GNU/Linux tools, i.e., LVM, and a careful definition of easily customizable virtual resource templates.

Even if the presented work is a successful proof-of-concept, many issues still have to be solved. In particular we have to assess the applicability of virtualized resources in HPC contexts, the payed overhead, and the possibility to extend the model to a mix of virtualized and physical resources according to the users needs.

We are working on solutions able to allow new communities wishing to use the grid to instantiate new grid infrastructure also for the non existing VOs. Moreover, we are also planning an evaluation of the impact on management operations and costs of our approach, in order to integrate a smart management of resources with the aim of providing energy savings, in a Green Computing perspective.

Acknowledgments. This work is part of the activities of a multidisciplinary group (GTT) that is responsible for the S.Co.P.E. infrastructure management. The work is also part of the activities carried out by GTT in the context of the Italian Grid Infrastructure (IGI).

References

1. Amazon: High Performance Computing (HPC) on AWS, <http://aws.amazon.com/hpc-applications/>
2. Andic, M., Dejan, Llorente, I.M., Montero, R.S.: Opennebula: A cloud management tool. *IEEE Internet Computing* 15(2), 11–14 (2011)
3. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review* 37, 164–177 (2003)
4. Barone, G., Bifulco, R., Boccia, V., Bottalico, D., Carracciuolo, L.: Toward a flexible, environmentally conscious, on demand high performance computing service. In: 2011 First International Conference on Data Compression, Communications and Processing (CCP), pp. 136–138 (June 2011)

5. Childs, S., Coghlan, B., McCandless, J.: Dynamic Virtual Worker Nodes in a Production Grid. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA 2006 Ws. LNCS, vol. 4331, pp. 417–426. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11942634_44
6. CloudFlu: CloudFlu - HPC cloud computing for OpenFOAM (R) users, <http://sourceforge.net/projects/cloudflu/>
7. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.* 15(3), 200–222 (2001), <http://dx.doi.org/10.1177/109434200101500302>
8. Laure, E., et al.: Programming the Grid with gLite. *Computational Methods in Science and Technology* 12(1), 33–45 (2006)
9. Loomis, C., Airaj, M., Bégin, M., Floros, E., Kenny, S., O’Callaghan, D.: StratusLab Cloud Distribution. In: Petcu, D., Vázquez-Poletti, J. (eds.) European Research Activities in Cloud Computing, pp. 260–282. Cambridge Scholars Publishing (2012)
10. Mell, P., Grance, T.: The NIST definition of Cloud Computing
11. Memon, M., Nagy, Z., Yen, E., Koeroo, O.: Virtualization and Cloud Computing Task Force Report V.0.7 (2012), <http://cdsweb.cern.ch/record/1359910/files/EMIVirtCloudReport-v0.7.doc>
12. Merola, L.: The S.Co.P.E. Project. In: Proceedings of the Final Workshop of Grid Projects of the Italian National Operational Programme 2000–2006 Call 1575, pp. 18–35. Consorzio COMETA (2009)
13. OpenFOAM-Foundation: OpenFOAM, <http://www.openfoam.com/>
14. Salomoni, D., Italiano, A., Ronchieri, A.: WNDeS, a tool for integrated Grid and Cloud access and computing farm virtualization. In: Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP 2010), pp. 18–35 (2011)
15. Tusa, F., Paone, M., Villari, M., Puliafito, A.: Clever: A cloud cross-computing platform leveraging grid resources. In: UCC, pp. 390–396. IEEE Computer Society (2011), <http://dblp.uni-trier.de/db/conf/ucc/ucc2011.html#TusaPVP11>

Author Index

- Abdelfattah, Ahmad 207
Albornoz, V.M. 196
Aldinucci, Marco 47, 381
Alexander, Michael 538
Aloisio, Giovanni 295
Alvarez-Mesa, Mauricio 155
Alves, Albano 112
Antoniu, Gabriel 36
Anzt, Hartwig 145
Archetti, Francesco 248
Armstrong, Django 567
Arnold, Dorian 484
Aulagnon, Charles 395
Ayguadé, Eduard 414
Ayoub, Raid 557
- Badia, Rosa M. 100
Baity-Jesi, M. 528
Bala, Piotr 263
Baniasadi, Amirali 518
Baños, R.A. 528
Barone, G.B. 577
Bartzas, Alexandros 507
Benner, Peter 102
Benoit, Anne 57
Bifulco, R. 577
Blanco, Héctor 196
Bland, Wesley 499
Bleda, Marta 273
Boccia, V. 577
Borcz, Marcelina 263
Bottalico, D. 577
Bouchenak, Sara 3
Bresnahan, John 36
Breuer, Stefan 370
Brightwell, Ron 484
Brorsson, Mats 133, 357
Buß, Matthias 370
- Campa, Sonia 47, 381
Cannataro, Mario 217
Canonico, R. 577
Carpen-Amarie, Alexandra 36
Carracciulo, L. 577
- Castaldi, Davide 248
Chi, Chi Ching 155
Copie, Adrian 24
Coqblin, Mathias 57
Costan, Alexandru 1
Cruz, A. 528
- Danelutto, Marco 47, 368
Das, Sajal K. 89
da Silva, Rafael Ferreira 79
Demeshko, Irina 175
Desprez, Frédéric 34
Dhiman, Gaurav 557
Dichev, Kiril 185
Dimopoulos, Nikitas J. 518
Diniz, Pedro C. 496
Djemame, Karim 567
Dobre, Ciprian 1
Dongarra, Jack 145
Dopazo, Joaquín 273
Duato, José 327
Duran, Alejandro 414
- Eckert, Wieland 123
Economakos, George 507
Elmroth, Erik 567
Espling, Daniel 567
Esteve, Albert 317
Ezzatti, Pablo 102
- Fernandez, L.A. 528
Ferreira, Kurt B. 484
Fllich, José 317
Fortiș, Teodor-Florin 24
Foshati, Amin 285
- Galgonek, Jakub 228, 238
Gil-Narvion, J.M. 528
Giménez, Judit 414
Glatard, Tristan 79
Gómez, María E. 305
Gómez, Maria Engracia 317
González, Cristina Y. 273
Gonzalez-Velez, H. 368
Gordillo-Guerrero, A. 528

- Gorlatch, Sergei 370
 Grzybowski, Tomasz 263
 Guidetti, M. 528
 Guirado, Fernando 196
- Hager, Georg 393, 451
 Hammond, K. 368
 Hannig, Frank 123
 Hasib, Abdullah Al 337
 Hast, Anders 505
 Hernández, Carles 327
 Heuveline, Vincent 145
 Hoksza, David 228, 238
 Hollingsworth, Jeffrey K. 404
 Hudzia, Benoit 539
 Hukerikar, Saurabh 496
- Ibtesham, Dewan 484
 Ince, Tugrul 404
 Iñiguez, D. 528
- Jooya, Ali 518
 Juurlink, Ben 155
- Kant, Krishna 89
 Keahey, Kate 36
 Keyes, David 207
 Khunjush, Farshad 285
 Kilpatrick, Peter 47, 381
 Kjeldsberg, Per Gunnar 337
 Kluszczyński, Rafal 263
 Kolano, Paul Z. 463
 Kontorinis, Vasileios 557
 Körner, Mario 123
 Koziris, Nectarios 548
 Krammer, Bettina 391
 Krishnaswamy, Ruby 13
 Kruliš, Martin 238
- Labarta, Jesús 414
 Lastovetsky, Alexey 185
 Leangsuksun, Chokchai (Box) 461, 474
 Lérida, Josep Lluís 196
 Llort, Germán 414
 Lorente, Vicent 347
 Ltaief, Hatem 207
 Lucas, Robert F. 496
- Maccagnola, Daniele 248
 Maiorano, A. 528
- Mantovani, F. 528
 Mari, Daniela 248
 Marinari, E. 528
 Marozzo, Fabrizio 220
 Martin-Guillerez, Damien 395
 Martin-Mayor, V. 528
 Martorell, Xavier 414
 Maruyama, Naoya 175
 Matsuoka, Satoshi 175
 Mayr, Georg 68
 Medina, Ignacio 273
 Membarth, Richard 123
 Mirto, Maria 295
 Momcilovic, Svetislav 165
 Monforte-García, J. 528
 Monnet, Sébastien 13
 Morar, Gabriela Andreea 68
 Muddukrishna, Ananya 357
 Muñoz-Sudupe, A. 528
 Munteanu, Victor Ion 24
- Nagel, Wolfgang E. 429
 Nanos, Anastassios 538, 548
 Nassar, Raja F. 474
 Natvig, Lasse 337
 Navarro, D. 528
 Nicod, Jean-Marc 57
- Ostermann, Simon 68
- Panda, Dhabaleswar K. (DK) 439
 Parisi, G. 528
 Passante, Marco 295
 Păun, Mihaela 474
 Perez-Gaviro, S. 528
 Petit, Salvador 305
 Philippe, Laurent 57
 Pina, António 112
 Pivanti, M. 528
 Podobas, Artur 133, 357
 Prodan, Radu 68
- Quintana-Ortí, Enrique S. 102
- Raj, Mayank 89
 Rao, Nageswara S.V. 494
 Rehn-Sonigo, Veronika 57
 Remón, Alfredo 102

- Ricci-Tersenghi, F. 528
Riesen, Rolf 484
Robles, Antonio 317
Rodrigo, Samuel 307
Roma, Nuno 165
Rosing, Tajana Simunic 557
Rué, François 395
Rufino, José 112
Ruiz-Lorenzo, J. 528
- Sadler, Chris 557
Sahuquillo, Julio 305, 347
Salavert, Francisco 273
Sánchez, Rubén 273
Sangroya, Amit 3
Santos, Luís Paulo 112
Schifano, S.F. 528
Schüller, Felix 68
Scott, Stephen L. 461
Sem-Jacobsen, Frank Olaf 307
Sens, Pierre 13
Seoane, B. 528
Serrano, Damián 3
Servat, Harald 414
Shribman, Aidan 539
Silla, Federico 327
Silvestre, Guthemberg 13
Skeie, Tor 307
Skonieczna, Katarzyna 263
Skopal, Tomáš 228
Soler, María 317
Sousa, Leonel 165
- Steuwer, Michel 370
Subramoni, Hari 439
- Talia, Domenico 34, 220
Tarancon, A. 528
Teich, Jürgen 123
Tellez, P. 528
Teruel, Xavier 414
Thanakornworakij, Thanadech 474
Tomita, Hirofumi 175
Tomov, Stanimire 145
Tordsson, Johan 567
Torquati, Massimo 47, 381
Trahay, François 395
Treibig, Jan 451
Tripiccione, R. 528
Trunfio, Paolo 220
Tullsen, Dean 557
- Vienne, Jerome 439
Vlassov, Vladimir 133, 357
- Wagner, Michael 429
Wang, Biao 155
Weidendorfer, Josef 505
Weiss, Jan-Philipp 505
Wellein, Gerhard 451
- Yayhapour, Ramin 34
Yllanes, D. 528
- Zanetti, Gianluigi 538
Zhang, Liuyi 557