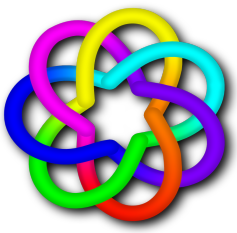
Conferences in Research and Practice in Information Technology

Volume 152

Parallel and Distributed Computing 2014

Australian Computer Science Communications, Volume 36, Number 6



Parallel and Distributed Computing 2014

Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing (AusPDC 2014), Auckland, New Zealand, 20 - 23 January 2014

Bahman Javadi and Saurabh Kumar Garg, Eds.

Volume 152 in the Conferences in Research and Practice in Information Technology Series.
Published by the Australian Computer Society Inc.

Published in association with the ACM Digital Library.

Parallel and Distributed Computing 2014. Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing (AusPDC 2014), Auckland, New Zealand, 20 - 23 January 2014

Conferences in Research and Practice in Information Technology, Volume 152.

Editors:

Bahman Javadi
School of Computing, Engineering and Mathematics
University of Western Sydney
Penrith, NSW 2751
Australia
Email: b.javadi@uws.edu.au

Saurabh Kumar Garg
IBM Research, Australia
Melbourne Research Laboratory
Victoria 3053
Australia
Email: skgarg@au1.ibm.com

Series Editors:
Vladimir Estivill-Castro, Griffith University, Queensland
Simeon J. Simoff, University of Western Sydney, NSW
Email: crpit@scem.uws.edu.au

Publisher: Australian Computer Society Inc.
PO Box Q534, QVB Post Office
Sydney 1230
New South Wales
Australia.

Conferences in Research and Practice in Information Technology, Volume 152.
ISSN 1445-1336.
ISBN 978-1-921770-34-0.

Document engineering, January 2014 by CRPIT
On-line proceedings, January 2014 by the University of Western Sydney
Electronic media production, January 2014 by the AUT University

Table of Contents

Contributed Papers

Preface

Parallel and distributed computing has played a key role in enabling execution of several scientific applications over the past years. With advances in technology, it has changed its scope from small clusters of workstations to very large-scale datacenters, which provide Cloud computing services. This proceedings volume presents some of the current research in this area that has contributed to the 12th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2014), held on 20-23 January 2014 in Auckland, New Zealand in conjunction with the Australasian Computer Science Week (ACSW 2014). In 2010, Australasian Symposium on Grid Computing and e-Research (AusGrid) was broadened to include all aspects of parallel and distributed computing and hence was called the Australasian Symposium on Parallel and Distributed Computing. Following a couple of successful events, AusPDC has become the flagship symposium for Grid, Cloud, Cluster, and Distributed Computing research in Australia and New Zealand.

Submissions were received from Australia, New Zealand, India, France, Japan, Taiwan, UK and US. The full version of each paper was carefully reviewed by at least two referees, and evaluated according to its originality, correctness, readability and relevance. A total of 7 papers (6 regular papers and one short paper) out of 14 submissions were accepted to be presented at the conference. The accepted papers cover topics from Cloud computing, Big Data, system security, GPU computing, and parallel processing systems. In addition to the technical papers, we are delighted to welcome an invited talk given by Professor Jemal Abawajy from Deakin University, Australia.

We are very thankful to the Program Committee members, and external reviewers for their outstanding and timely work, which was invaluable for taking the quality of this year's program to such a high level. We also wish to acknowledge the efforts of the authors who submitted their papers and without whom this conference would have not been possible. Due to the competitive selection process, several strong papers could not be included in the program. We sincerely hope that prospective authors will continue to view the AusPDC symposium series as the premiere venue in the field for disseminating their work and results. We would like to acknowledge the leadership and untiring efforts of the conference General Chairs, A./Professor Tony Clear and Dr. Russel Pears and the guidance provided by the steering committee, in particular Professor Rajkumar Buyya, A./Professor Jinjun Chen, and Dr. Rajiv Ranjan.

We are grateful to ACSW Organizing Committee and Professor Simeon Simoff from UWS representing CRPIT for his assistance in the production of the proceedings. We would like to thank IBM Research, Australia for their support and sponsor the Best Paper Award. Thanks to the School of Computing, Engineering, and Mathematics of University of Western Sydney for web support, advertising and refereeing for the conference.

Bahman Javadi
University of Western Sydney

Saurabh Kumar Garg
IBM Research, Australia

AusPDC 2014 Programme Chairs
January 2014

Programme Committee

Chairs

Bahman Javadi, University of Western Sydney, Australia
Saurabh Kumar Garg, IBM Research, Australia

Members

Jemal Abawajy, Deakin University, Australia
Ermyas Abebe, IBM Research, Australia
David Abramson, Monash University, Australia
Anton Beloglazov, IBM Research, Australia
Peter Bertok, RMIT, Australia
Borzoo Bonakdarpour, University of Waterloo, Canada
Rajkumar Buyya, The University of Melbourne, Australia
Geffrey Fox, Indiana University, USA
Andrzej Goscinski, Deakin University, Australia
Kenneth Hawick, Massey University, New Zealand
Michael Hobbs, Deakin University, Australia
Zhiyi Huang, Otago University, New Zealand
Wayne Kelly, Queensland University of Technology, Australia
Kevin Lee, Murdoch University, Australia
Young Choon Lee, The University of Sydney, Australia
Laurent Lefevre, INRIA, University of Lyon, France
Weifa Liang, Australian National University, Australia
Farhad Mehdipour, Kyushu University, Japan
Paul Roe, Queensland University of Technology, Australia
Jun Shen, University of Wollongong, Australia
Weisheng Si, University of Western Sydney, Australia
Gaurav Singh, CSIRO Mathematical and Information Sciences, Australia
Richard Sinnott, The University of Melbourne, Australia
Kurt Vanmechelen, University of Antwerp, Belgium
Andrew Wendelborn, University of Adelaide, Australia
Yulei Wu, Chinese Academy of Sciences, China
Yang Xiang, Deakin University, Australia
Jingling Xue, University of New South Wales, Australia
Jun Yan, University of Wollongong, Australia
Yun Yang, Swinburne University of Technology, Australia
Albert Zomaya, The University of Sydney, Australia

Steering Committee

Prof. David Abramson, Monash University, Australia
Prof. Rajkumar Buyya, University of Melbourne, Australia
A./Prof. Jinjun Chen (Vice Chair), University of Technology Sydney, Australia
Dr. Paul Coddington, University of Adelaide, Australia
Prof. Andrzej Goscinski (Chair), Deakin University, Australia
Prof. Kenneth Hawick, Massey University, New Zealand
Prof. John Hine, Victoria University of Wellington, New Zealand
Dr. Rajiv Ranjan, CSIRO ICT Centre, Australia
Dr. Wayne Kelly, Queensland University of Technology, Australia
Prof. Paul Roe, Queensland University of Technology, Australia
Dr. Andrew Wendelborn, University of Adelaide, Australia
Dr. Bahman Javadi, University of Western Sydney, Australia
Dr. Saurabh Kumar Garg, IBM Research, Australia

Organising Committee

Chairs

Tony Clear and Russel Pears

Venue

Tony Clear

Communications

Russel Pears, Hui Ling Tan, Melanie Curry-Irons and Ryan Butler

Finance

Alison Clear and Eva Ihaia

Sponsorship

Stephen Thorpe

Operations

Adam Winship and Eva Ihaia

Programme, proceedings and booklet

Alison Clear

Catering and registration

AUT Hospitality Services

Welcome from the Organising Committee

On behalf of the Organising Committee, it is our pleasure to welcome you to Auckland and to the 2014 Australasian Computer Science Week (ACSW 2014). Auckland is New Zealand's largest urban area with a population of nearly one and a half million people. As the centre of commerce and industry, Auckland is the most vibrant, bustling and multicultural city in New Zealand. With the largest Polynesian population in the world, this cultural influence is reflected in many different aspects of city life. ACSW 2014 will be hosted at the City Campus of Auckland University of Technology (AUT), which is situated just up from the Town Hall and the Auckland central business district. ACSW is the premier event for Computer Science researchers in Australasia. ACSW2014 consists of conferences covering a wide range of topics in Computer Science and related areas, including:

- Australasian Computer Science Conference (ACSC) (Chaired by Bruce Thomas and Dave Parry)
- Australasian Computing Education Conference (ACE) (Chaired by Jacqueline Whalley and Daryl D'Souza)
- Australasian Information Security Conference (AISC) (Chaired by Udaya Parampalli and Ian Welch)
- Australasian User Interface Conference (AUIC) (Chaired by Burkhard C. Wünsche and Stefan Marks)
- Australasian Symposium on Parallel and Distributed Computing (AusPDC) (Chaired by Bahman Javadi and Saurabh Kumar Garg)
- Australasian Workshop on Health Informatics and Knowledge Management (HIKM) (Chaired by James Warren)
- Asia-Pacific Conference on Conceptual Modelling (APCCM) (Chaired by Georg Grossmann and Motoshi Saeki)
- Australasian Web Conference (AWC) (Chaired by Andrew Trotman)

This year reflects an increased emphasis for ACSW on community building. Complementing these published technical volumes therefore, ACSW also hosts two doctoral consortia and a number of associated workshops, including those for the Heads and Professors of Computer Science, plus for the first time the 'Australasian Women in Computing Celebration'. Naturally in additional to the technical program, there are a range of events, which aim to provide the opportunity for interactions among our participants. A welcome reception will be held in the atrium of the award winning newly built Sir Paul Reeves Building, which has integrated the city campus as a hub for student activity and provides a wonderful showcase for this year's ACSW. The conference banquet will be held on campus in one of the reception rooms in this impressive complex.

Organising a multi-conference event such as ACSW is a challenging process even with many hands helping to distribute the workload, and actively cooperating to bring the events to fruition. This year has been no exception. We would like to share with you our gratitude towards all members of the organising committee for their combined efforts and dedication to the success of ACSW2014. We also thank all conference co-chairs and reviewers, for putting together the conference programs which are the heart of ACSW, and to the organisers of the symposia, workshops, poster sessions and accompanying conferences. Special thanks to Alex Potanin, as the steering committee chair who shared valuable experiences in organising ACSW and to John Grundy as chair of CoRE for his support for the innovations we have introduced this year. We'd also like to thank Hospitality Services from AUT, for their dedication and their efforts in conference registration, venue, catering and event organisation. This year we have secured generous support from several sponsors to help defray the costs of the event and we thank them for their welcome contributions. Last, but not least, we would like to thank all speakers, participants and attendees, and we look forward to several days of stimulating presentations, debates, friendly interactions and thoughtful discussions.

We hope your stay here will be both rewarding and memorable, and encourage you to take the time while in New Zealand to see some more of our beautiful country.

Tony Clear
Russel Pears
School of Computer & Mathematical Sciences

ACSW2014 General Co-Chairs
January, 2014

CORE - Computing Research & Education

CORE welcomes all delegates to ACSW2014 in Auckland. CORE, the peak body representing academic computer science in Australia and New Zealand, is responsible for the annual ACSW series of meetings, which are a unique opportunity for our community to network and to discuss research and topics of mutual interest. The component conferences of ACSW have changed over time with additions and subtractions ACSC, ACE, AISC, AUIC, AusPDC, HIKM, ACDC, APCCM, CATS and AWC have now been joined by the Australasian women in computing celebration (AWIC), two doctoral consortia (ACDC and ACE-DC)and an Australasian Early Career Researchers Workshop (AECRW) which reflect the evolving dimensions of ACSW and build on the diversity of the Australasian computing community.

In 2014, we have again chosen to feature a small number of keynote speakers from across the discipline: Anthony Robins (ACE), John Mylopolous (APCCM), and Peter Gutmann (AISC). I thank them for their contributions to ACSW2014. The efforts of the conference chairs and their program committees have led to strong programs in all the conferences, thanks very much for all your efforts. Thanks are particularly due to Tony Clear, Russel Pears and their colleagues for organising what promises to be a vibrant event. Below I outline some of COREs activities in 2012/13.

I welcome feedback on these including other activities you think CORE should be active in.

The major sponsor of Australian Computer Science Week:
- The venue for the annual Heads and Professors meeting
- An opportunity for Australian & NZ computing staff and postgrads to network and help develop their research and teaching
- Substantial discounts for attendees from member departments
- A doctoral consortium at which postgrads can seek external expertise for their research
- An Early Career Research forum to provide ECRs input into their development

Sponsor of several research, teaching and service awards:
- Chris Wallace award for Distinguished Research Contribution
- CORE Teaching Award
- Australasian Distinguished Doctoral Dissertation
- John Hughes Distinguished Service Award
- Various Best Student Paper awards at ACSW

Development, maintenance, and publication of the CORE conference and journal rankings. In 2013 this includes a new portal with a range of holistic venue information and a community update of the CORE 2009 conference rankings.

Input into a number of community resources and issues of interest:
- Development of an agreed national curriculum defining Computer Science, Software Engineering, and Information Technology
- A central point for discussion of community issues such as research standards
- Various submissions on behalf of Computer Science Departments and Academics to relevant government and industry bodies, including recently on Australian Workplace ICT Skills development, the Schools Technology Curriculum and the Mathematics decadal plan

Coordination with other sector groups:
- Work with the ACS on curriculum and accreditation
- Work with groups such as ACDICT and government on issues such as CS staff performance metrics and appraisal, and recruitment of ?students into computing
- A member of CRA (Computing Research Association) and Informatics Europe. These organisations are the North American and European equivalents of CORE.
- A member of Science & Technology Australia, which provides eligibility for Science Meets Parliament and opportunity for input into government policy, and involvement with Science Meets Policymakers

A new Executive Committee from 2013 has been looking at a range of activities that CORE can lead or contribute to, including more developmental activities for CORE members. This has also included a revamp of the mailing lists, creation of discussion forums, identification of key issues for commentary and lobbying, and working with other groups to attract high aptitude students into ICT courses and careers. Again, I welcome your active input into the direction of CORE in order to give our community improved visibility and impact.

CORE's existence is due to the support of the member departments in Australia and New Zealand, and I thank them for their ongoing contributions, in commitment and in financial support. Finally, I am grateful to all those who gave their time to CORE in 2013, and look forward to the continuing shaping and development of CORE in 2014.

John Grundy

President, CORE
January, 2014

ACSW Conferences and the Australian Computer Science Communications

The Australasian Computer Science Week of conferences has been running in some form continuously since 1978. This makes it one of the longest running conferences in computer science. The proceedings of the week have been published as the *Australian Computer Science Communications* since 1979 (with the 1978 proceedings often referred to as *Volume 0*). Thus the sequence number of the Australasian Computer Science Conference is always one greater than the volume of the Communications. Below is a list of the conferences, their locations and hosts.

2015. Volume 37. Host and Venue - University of Western Sydney, NSW.

2014. **Volume 36. Host and Venue - AUT University, Auckland, New Zealand**.

2013. Volume 35. Host and Venue - University of South Australia, Adelaide, SA.
2012. Volume 34. Host and Venue - RMIT University, Melbourne, VIC.
2011. Volume 33. Host and Venue - Curtin University of Technology, Perth, WA.
2010. Volume 32. Host and Venue - Queensland University of Technology, Brisbane, QLD.
2009. Volume 31. Host and Venue - Victoria University, Wellington, New Zealand.
2008. Volume 30. Host and Venue - University of Wollongong, NSW.
2007. Volume 29. Host and Venue - University of Ballarat, VIC. First running of HDKM.
2006. Volume 28. Host and Venue - University of Tasmania, TAS.
2005. Volume 27. Host - University of Newcastle, NSW. APBC held separately from 2005.
2004. Volume 26. Host and Venue - University of Otago, Dunedin, New Zealand. First running of APCCM.
2003. Volume 25. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Adelaide Convention Centre, Adelaide, SA. First running of APBC. Incorporation of ACE. ACSAC held separately from 2003.
2002. Volume 24. Host and Venue - Monash University, Melbourne, VIC.
2001. Volume 23. Hosts - Bond University and Griffith University (Gold Coast). Venue - Gold Coast, QLD.
2000. Volume 22. Hosts - Australian National University and University of Canberra. Venue - ANU, Canberra, ACT. First running of AUIC.
1999. Volume 21. Host and Venue - University of Auckland, New Zealand.
1998. Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.
1997. Volume 19. Hosts - Macquarie University and University of Technology, Sydney. Venue - Sydney, NSW. ADC held with DASFAA (rather than ACSW) in 1997.
1996. Volume 18. Host - University of Melbourne and RMIT University. Venue - Melbourne, Australia. CATS joins ACSW.
1995. Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.
1994. Volume 16. Host and Venue - University of Canterbury, Christchurch, New Zealand. CATS run for the first time separately in Sydney.
1993. Volume 15. Hosts - Griffith University and Queensland University of Technology. Venue - Nathan, QLD.
1992. Volume 14. Host and Venue - University of Tasmania, TAS. (ADC held separately at La Trobe University).
1991. Volume 13. Host and Venue - University of New South Wales, NSW.
1990. Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).
1989. Volume 11. Host and Venue - University of Wollongong, NSW.
1988. Volume 10. Host and Venue - University of Queensland, QLD.
1987. Volume 9. Host and Venue - Deakin University, VIC.
1986. Volume 8. Host and Venue - Australian National University, Canberra, ACT.
1985. Volume 7. Hosts - University of Melbourne and Monash University. Venue - Melbourne, VIC.
1984. Volume 6. Host and Venue - University of Adelaide, SA.
1983. Volume 5. Host and Venue - University of Sydney, NSW.
1982. Volume 4. Host and Venue - University of Western Australia, WA.
1981. Volume 3. Host and Venue - University of Queensland, QLD.
1980. Volume 2. Host and Venue - Australian National University, Canberra, ACT.
1979. Volume 1. Host and Venue - University of Tasmania, TAS.
1978. Volume 0. Host and Venue - University of New South Wales, NSW.

Conference Acronyms

ACDC	Australasian Computing Doctoral Consortium
ACE	Australasian Computing Education Conference
ACSC	Australasian Computer Science Conference
ACSW	Australasian Computer Science Week
ADC	Australasian Database Conference
AISC	Australasian Information Security Conference
APCCM	Asia-Pacific Conference on Conceptual Modelling
AUIC	Australasian User Interface Conference
AusPDC	Australasian Symposium on Parallel and Distributed Computing (replaces AusGrid)
AWC	Australasian Web Conference
CATS	Computing: Australasian Theory Symposium
HIKM	Australasian Workshop on Health Informatics and Knowledge Management

Note that various name changes have occurred, which have been indicated in the Conference Acronyms sections in respective CRPIT volumes.

ACSW and AusPDC 2014 Sponsors

We wish to thank the following sponsors for their contribution towards this conference.

Host Sponsor

Auckland University of Technology,
www.aut.ac.nz

Platinum Sponsor

DATACOM,
www.datacom.com.au

Gold Sponsor

The University of Auckland,
www.auckland.ac.nz

Silver Sponsors

COLAB - AUT Design+Creative Technologies,
colab.aut.ac.nz

Institute of IT Professionals, New Zealand,
www.iitp.org.nz

CORE - Computing Research and Education,
www.core.edu.au

Bronze Sponsor

SERL - AUT Software Engineering Research Laboratory,
www.serl.aut.ac.nz

Australian Computer Society,
www.acs.org.au

Publication Sponsor

University of Western Sydney,
www.uws.edu.au

Sponsors of AusPDC

IBM Research – Australia,
www.research.ibm.com/labs/australia/index.shtml

Contributed Papers

A Load-Balanced MapReduce Algorithm for Blocking-based Entity-resolution with Multiple Keys

Sue-Chen Hsueh[1], Ming-Yen Lin[2]*, and Yi-Chun Chiu[2]
[1]Dept. of Information Management, Chaoyang University of Technology, Taichung, Taiwan
[2]Dept. of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan
schsueh@cyut.edu.tw; linmy@mail.fcu.edu.tw; ojean913@gmail.com

Abstract

Entity resolution (ER), which detects records referring to the same entity across data sources, is a long-lasting challenge in database management research. The sheer volume of data collections today calls for the need of a blocking-based ER algorithm using the MapReduce framework for cloud computing. Most studies on blocking-based ER assume that only one blocking key is associated with an entity. An entity in reality may have multiple blocking keys in some applications. When the entities have a number of blocking keys, ER can be more efficient since two entities can form a similar pair only if they share several common keys. Therefore, we propose a MapReduce algorithm to solve the ER problem for a huge collection of entities with multiple keys. The algorithm is characterized in the combination-based blocking and the load-balanced matching. The combination-based blocking utilizes the multiple keys to sort out necessary entity pairs for future matching. The load-balanced matching evenly distributes the required similarity computations to all the reducers in the matching step so as to remove the bottleneck of skewed matching computations for a single node in a MapReduce framework. Our experiments using the well-known CiteSeerX digital library show that the proposed algorithm is both efficient and scalable.

Keywords: Entity resolution, cloud computing, MapReduce, load-balance.

1 Introduction

Entity resolution (abbreviated as ER), also known as data matching, de-duplication, identity resolution, or record linkage, is to identify all the manifestations referring to the same entity (Hernández and Stolfo 1995; Brizan and Tansel 2006; Chaudhuri, Ganti, and Kaushik 2006; Elsayed et al. 2008; Vernica et al. 2010; Zhang et al. 2010). ER is crucial for data integration and has been applied in many applications including price comparisons, citation matching (Pasula et al. 2012), etc. Take a price-comparison website for example, to provide all the prices of a product crawled from different web-pages, the website needs to identify whether the two product-pages from two product websites referring to the same product. Given n_1 product-pages in website P and n_2 product-pages in website Q, n_1*n_2 similarity computations are required to identify the pages for the same products by entity resolution. The total number of similarity computations can be massive, considering the scale of web-pages nowadays.

In general, a *blocking-based ER* is used to accelerate the similarity computations. Each entity in the blocking-based ER has an attribute or a specific value computed by hashing or other means called *blocking key*. The blocking-based ER comprises a blocking step and a matching step. The blocking step partitions all the entities in the dataset by blocking keys into several groups called *blocks*, where the entities in a block have the same blocking key. After that, the matching step performs similarity computations for all the entity pairs in each block and determines the similarities between entities within each block. The result is a set of all the entity pairs of high similarity. The blocking key in fact reduces the total number of similarity computations required in the blocking-based ER.

Most of the studies on blocking-based ER assume that there is only one blocking key associated with an entity. In reality, an entity may have several blocking keys. An entity can be a record in a database, a web-page in a web-site, an article in a research repository, etc. A product page may be classified to multiple categories so as to have several attribute keys; an article in DBLP or CiteSeerX generally contains many keywords or index terms. In addition, the titles and some attributes like authors of the articles can serve as blocking keys, too. When the entities have a number of blocking keys, blocking-based ER can be performed more efficient since two entities can be a similar pair only if they share several common keys.

Nevertheless, the blocking step of ER will become more time-consuming if multiple blocking keys are treated as many individual blocking keys, which consideration is adopted in most ER algorithms today. Traditional solution generates a pair for further matching if two entities share a common blocking key. Because there are more than one key associated with an entity now, the possibility of sharing common keys with other entities, which also contain many keys, is greatly increased. Consequently, the number of entity pairs generated for matching will be greatly increased. We consider that blocking keys represent specific features of entities, multiple blocking keys can be used in the blocking step to sort out the entities need to be matched so that the matching step can be accelerated.

A typical ER problem is the pairwise document similarity (Baraglia et al. 2010; Lin 2009; Elsayed et al. 2008), which detects similar documents in a collection like DBLP, CiteSeerX, or Google Scholar. The collection of articles is continuously growing as research articles keep

appending in a surprising speed. The ER process performed by a single machine suffers from the vast amount of the increasing data. The total number of similarity computations for the potential pairs becomes very huge. For example, 1.4 million publication records in the CiteSeerX collection will require 979 billion similarity computations. The problem is exacerbated if the data size accumulates up to terabytes or petabytes. Thus, it is necessary to devise a blocking-based ER algorithm using the MapReduce framework (Dean and Ghemawat 2004) in an open architecture like Hadoop (Hadoop 2012).

Although the blocking-based ER using the MapReduce framework may solve the problem of huge dataset size, the process may still suffer from the imbalance of key distributions. Commonly, the map phase and the reduce phase are the two core processes of a MapReduce program. An input record is a form of (key, value) pair and the key may come from a different domain. The process responsible for the map function is referred to a *mapper* and that for the reduce function is referred to a *reducer* for convenience. A mapper reads one block of (key, value) pairs, and produces a list of intermediate (key', value') pairs after processing. Each (key', value') pair having the same key' will be sent to the same reducer. A reducer accepts the intermediate key' with corresponding list of value's, processes and generates the output. A default hash partitioning is used in the MapReduce framework to partition the intermediate (key', value') pairs to the reducers. Each reducer might be responsible for its respective 100 (key', value') pairs when there are 10000 keys and 100 reducers. If certain keys are more popular than the others, a large number of (key', value') pairs will be sent to some reducer so that the ER task cannot finish until this reducer accomplishes matching. In fact, the bottleneck of the ER process is on the reducer having the largest number of assigned (key', value') pairs. To our knowledge, few studies have taken account of the load-balance problem among reducers, especially for the ER problem. The efficiency of the whole ER process can be remarkably improved when the working loads of the reducers in the matching step are balanced.

Therefore, in this paper, we propose a MapReduce algorithm to solve the ER problem for a huge collection of entities with multiple keys. The algorithm features in the combination-based blocking and the load-balanced matching. The combination-based blocking utilizes the multiple keys to filter out unnecessary entity pairs. The load-balanced matching evenly distributes the required similarity computations to all the reducers in the matching step. Our experiments using CiteSeerX show that the proposed algorithm is efficient and scales up linearly with respect to the dataset size.

The rest of the paper is organized as follows. Section 2 defines our problem. Related works are briefly reviewed in Section 3. Section 4 presents the proposed algorithm. The experimental results are given in Section 5. Section 6 concludes this study.

2 Problem Definition

A dataset D = $\{E_1, E_2, \ldots, E_m\}$ contains m entities, where an entity E_i ($1 \le i \le m$) has $|E_i|$ blocking keys. The blocking keys of E_i is represented by $B_{E_i} = \{k_1, k_2, \ldots, k_{|E_i|}\}$ and $|E_i| > 1$. Let the minimum number of blocking keys of the entities in D be kc, $kc > 1$. Given a similarity measure and a similarity threshold θ, the ER problem is to find out all the *similar* entity pairs in D. An entity pair (E_i, E_j) is considered *similar* if $\text{sim}(E_i, E_j) \ge \theta$, where $\text{sim}(E_i, E_j)$ is the similarity value between E_i and E_j computed using the similarity measure. Particularly, the characteristics of blocking keys implies, in our problem, that $\text{sim}(E_i, E_i) < \theta$ if $|B_{E_i} \cap B_{E_i}| < kc$. That is, E_i and E_j cannot be similar if the number of common blocking keys between them is less than kc. Accordingly, the similarity computation $\text{sim}(E_i, E_j)$ can be eliminated in the ER process if $|B_{E_i} \cap B_{E_j}| < kc$.

A typical similarity measure between E_i and E_j is the *Jaccard similarity coefficient*, which is defined as the size of their intersection divided by the size of their union. Assume the entities are text research articles, the Jaccard similarity coefficient can be calculated by the number of common terms divided by the total number of terms in the two articles. The blocking keys of the entities can be title words or keywords of the articles.

For example, given D = $\{E_1, E_2, \ldots, E_{100}\}$ and $kc = 2$. We have 100 entities and the minimum number of blocking keys of the entities in D is 2. Assume E_1 has 4 blocking keys B_{E_1} = {a, c, d, e}, E_2 has 3 blocking keys B_{E_2} = {a, c, e}, and E_3 has 2 blocking keys B_{E_3} = {d, e}, etc. Then $|E_1| = 4$, $|E_2| = 3$, and $|E_3| = 2$; $B_{E_1} \cap B_{E_2}$ = {a, c, e}, $B_{E_1} \cap B_{E_3}$ = {d, e}, and $B_{E_2} \cap B_{E_3}$ = {e}. The similarity computations need to be performed on both (E_1, E_2) and (E_1, E_3) but not (E_2, E_3) since $|B_{E_2} \cap B_{E_3}| = 1 < kc$. If the similarity threshold $\theta = 0.8$, $\text{sim}(E_1, E_2) = 0.85$, and $\text{sim}(E_1, E_3) = 0.7$, entity pair ($E_1$, E_2) is a similar pair and is inserted into the answer set.

Note that the ER problem definition is the same as others. Nevertheless, our problem emphasizes on the variable number of blocking keys in entities. Only entities having certain number of common keys are potentially similar.

3 Related Works

Although the ER problem is a long-existing problem, most solutions are not designed for extremely large collection of entities. Some algorithms have been presented for document-similarity computations (Baraglia et al. 2010) and blocking-based ER solution under the MapReduce framework (Kiefer et al. 2010; Kim and Shim 2012; Lu et al. 2012; Metwally and Faloutsos 2012; Vernica et al. 2010; Zhang et al. 2010). These works assume that there is only one key for an entity and use a map/reduce phase to handle the problem. These works design a map function for the blocking step and then a reduce function for the matching step. Some of the notable works includes sorted neighborhood (Kolb et al. 2011a) and load-balanced ER (Kolb et al. 2012a; Kolb et al. 2012b).

Sorted neighborhood is a blocking technique by sorting all entities according to blocking keys, assigning a window size w and comparing entities in the window while sliding. Some entities with similar but not same blocking keys might be compared for similarity evaluations. In (Kolb et al. 2011a; Kolb et al. 2011b), the sorted neighborhood is implemented under the MapReduce framework while

entities crossing boundaries have to be duplicated to avoid missing potential pairs due to reducers' limitations (Kolb et al. 2011a). Both JobSN using two MapReduce phases and RepSN using one MapReduce phase are presented. However, the problem of imbalanced loads among reducers in MapReduce is not mentioned.

BlockSplit and PairRange (Kolb et al. 2012b) consider the load-balancing problem using the MapReduce framework for single key ER. A block distribution matrix needs to be distributed using the distributedCache technique in MapReduce so that the entities can be evenly distributed to all the reducers. PairRange outperforms BlockSplit because PairRange guarantees that all reducers may receive the same number of entities (Kolb et al. 2012b).

The study in (Kolb et al. 2011a) indicates that multiple keys may exist in an entity. However, the multiple keys are treated as independent like several single keys so that duplicate distributions appear in the blocking step. The MultiRepSN (Kolb et al. 2011a) is proposed to overcome an entity with multiple blocking keys. Although reducer-loads are balanced, the algorithm might suffer from the problem of largely duplicated entity pairs.

Thus, the study (Kolb et al. 2013) presents an algorithm for ER with redundancy-free matching. The idea is to enumerate all candidate sets of entities with the index of the smallest common candidate sets. The mapper will emit (blocking keys, entity values) for the reducers. The entity value includes the entity and the smallest common blocking keys. Thus, Redundancy-Free Matching (Kolb et al. 2013; Kolb and Rahm 2013) may decrease the number of duplicate entity pairs. However, when a reducer receives an overwhelming entities, the execution time can be long. Therefore, the issue of load-balancing among reducers must be considered for an ER solution under the MapReduce framework with multiple keys.

4 The Proposed Algorithm

4.1 Overview of the Algorithm

The proposed algorithm for blocking-based ER utilizes the multiple blocking keys in each entity for an improved entity distribution in the blocking step. The distribution is more precise because an entity pair may exist in a block only when the number of common blocking keys between the pair exceeds certain threshold (i.e. *kc*). Because an entity may have more than *kc* keys, it needs to generate all the combinations of *kc* keys for potential key comparisons. The entity distribution procedure is called combination-based blocking in the proposed algorithm.

After the blocking step, we aim to balance the working load among all the reducers for similarity computations in the matching step. The idea is to obtain a statistics of the total number of computations required for all the blocks first. We then evenly partition the computations among all the reducers to avoid potential overload of a reducer due to skewed key distributions. The procedure is called load-balanced matching in the proposed algorithm. Both combination-based blocking and load-balanced matching are designed using the MapReduce framework. Therefore, the algorithm comprises two map/reduce phases: a map/reduce phase for combination-based blocking, followed by a map/reduce phase for load-balanced matching. An overview of the proposed algorithm is shown in Fig. 1.

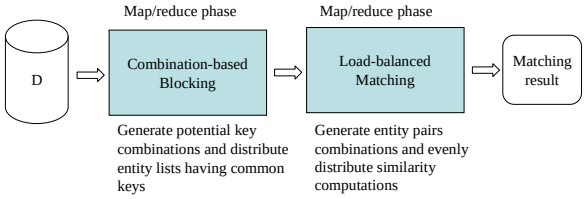


Fig. 1: An Overview of the Proposed Algorithm

Map(key = E_i, value = B_{Ei})

Input: P = a partition of entities in D; *kc* = minimum number of common keys

1. foreach $E_i \in$ P do
2. foreach $K_b = (a_1, a_2, \ldots, a_{kc})$, where $a_p, a_q \in B_{Ei} \wedge a_p \neq a_q \; \forall \; 1 \leq p, q \leq kc$
3. output <K_b, *i*> /* K_b is a *kc*-combination from B_{Ei}, *i* is the id of entity E_i */
4. end
5. end

Reduce(key = K_b, value = list of entity ids)

1. foreach key K_b do /* initial *entity_list* = ∅ */
2. foreach entity id *v* in the list of values do
3. add *v* to *entity_list*
4. end
5. if *entity_list*.count ≥ 2
6. output < K_b, *entity_list*>
7. endif
8. end

Fig. 2: Functions for Combination-based Blocking

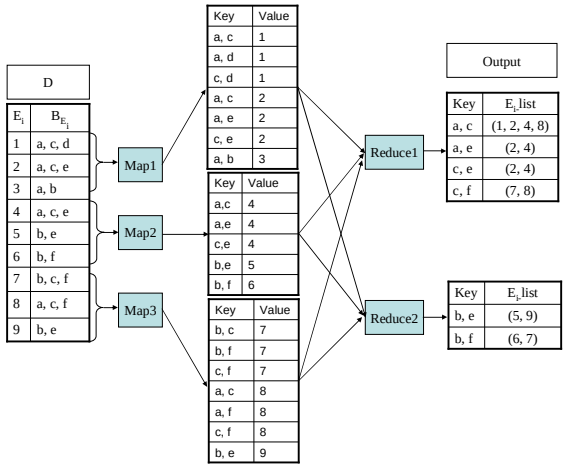


Fig. 3: An Example for Combination-based Blocking

4.2 Combination-based Blocking

Figure 2 shows the map/reduce functions in the proposed combination-based blocking procedure. A mapper receives its partition of entities in the dataset. Each input data is an entity and the associated blocking keys. The map function generates all the *kc*-combinations from the set of

blocking keys. Each combination is outputted with the id of the entity. For example, <(a,c), 2>, <(a,e), 2>, and <(c,e), 2> are outputted for entity E_2 with B_{E_2} = {a, c, e} when *kc* = 2; <(a, c, e), 2> is outputted for E_2 and <(a, c, d), 1>, <(a, c, e), 1>, <(a, d, e), 1>, and <(c, d, e), 1> are outputted for E_1 with B_{E_1} = {a, c, d, e} when *kc* = 3. In fact, E_1 and E_2 need to be compared since they have three common blocking keys when *kc* = 3. Thus, all the *kc*-combinations from B_{E_i} for entity E_i need to be generated.

The reduce function of the procedure further determines those entities having at least *kc* common keys to form a potential list for the matching step. The reducer would receive a *kc*-combination as the key and the list of entity ids having that combination. If the list contains only one entity for a certain key combination, the entity is eliminated from the matching step obviously. Therefore, we count the number of entities in the *entity_list* (*entity_list*.count) while adding entities of the same K_b to the associated *entity_list*. Only *entity_list* having two or more entities will be sent to the load-balanced matching.

An example showing the map/reduce functions with nine entities, three mappers, two reducers, and kc = 2 is displayed in Fig 3. Mapper1 generates <(a,c), 1>, <(a, d), 1>, <(c, d), 1>, and so on; Mapper2 generates <(a, c), 4>, <(a,e), 4> and so forth. A simple hash partitioning is adopted for the reducers. Assume Reducer1 accepts keys (a, c), (a, e), and so on. Hence, entity_list (1, 2, 4, 8) of the same key (a, c) will be sent to the next procedure, load-balanced matching.

4.3 Load-balanced Matching

The matching step has to compute similarities for the pairs of all the 2-combnations of each entity list generated by the blocking step. A common implementation of the matching procedure uses the default hash partitioning to pass the intermediate keys to reducers. Consequently, the reducer receiving long entity-lists suffer from the vast amount of similarity computations. A skewed key distribution also leads to imbalanced workloads for reducers. Hence, the matching step cannot finish until the reducer of the heaviest workload completes. Take the entity_lists in Fig. 3 for example, reducer1 produces 7 entity pairs (including (1,2), (1, 4), (1, 8), (2, 4), (2, 8), (4, 8), and (7,8)) but reducer2 produces only 2 pairs for similarity computations. The blocking-based ER can be completed earlier without unnecessary waiting if the workloads are balanced among reducers in the matching step.

Figure 4 shows the map/reduce functions in the proposed load-balanced matching procedure. Note that the *partition* function, which replaces the default hash partitioning, is particularly designed to evenly distribute the total number of comparisons to the total number of reducers (i.e. *numReduceTasks*). In order to average the workloads, we use a partition number *partno* to count and mark each input record (entity pair). The total number of entity pairs for matching is then obtained by the *partno*. Note that we prune the duplicate pair (2, 4) from keys (a, e), and (c, e) here. The mapper generates (*partno*, entity pair) as the (key, value) for the reducers but applying the designated partitioning for load balancing. Entity pairs are sent to all the receivers by round robin, as shown in the *partition* function in Fig. 4. The reducers then outputs the similar pairs if their similarity value is no less than the similarity threshold.

Figure 5 shows an example of load-balanced matching with two mappers and two reducers. The matching receives entity lists in Fig. 3.

A mapper generates the <*partno*, *entity pair*> by enumerating all the 2-combinations of an entity list with associated *partno*, which value is increased by 1 for each combination. The entity list (1, 2, 4, 8) will produce (1, 2), (1, 4), (1, 8), (2, 4), (2, 8), and (4, 8) entity pairs. Thus, <1, (1, 2)>, <2, (1, 4)>, <3, (1, 8)>, <4, (2, 4)>, <5, (2, 8)>, and <6, (4, 8)> are generated by mapper1; <7, (7, 8)>, <8, (5, 9)>, and <9, (6, 7)> are generated by mapper2. The partitioning determines which reducer to receive the <*partno*, *entity pair*> using the designed partition function. Because the total number of reducers is two, <*partno*, *entity pair*> having odd *partno* is processed by reducer1, and that having even *partno* is processed by reducer2. Assume that the similarity value of sim(E_1, E_8), sim(E_2,E_8), sim(E_6, E_7), and sim(E_4, E_8) is less than the similarity threshold. The matched result contains similar pairs (1, 2), (7, 8), (1, 4), (2, 4), and (5, 9). If we configure the MapReduce job with 3 reducers, the 9 computations will be evenly distributed to the 3 reducers.

Map(key = K_b , value = entity list E_i*_list*)
1. foreach entity_list E_i*_list* do /* generate all 2-combination entity pairs from E_i*_list* */
2. generate all 2-combination entity pairs *EPS* = { (*Ep*, *Eq*) | *Ep*, *Eq* ∈ E_i*_list* ∧ *Ep* ≠ *Eq* }
3. foreach entity pair (*Ep*, *Eq*) in *EPS* do /*initial partno = 1*/
4. partno++;
5. output <partno, (*Ep*, *Eq*) >
6. end
7. end

Partition(key = *partno*, value = entity pair (*Ep*, *Eq*) **)**
/* numReduceTasks is the number of reducers, defined by MapReduce configuration */
return (*partno*.hashCode() & Integer.MAX_VALUE) % numReduceTasks;

Reduce(key = *partno*, value = entity pair (*Ep*, *Eq*) **)**
1. foreach key *partno* do
2. foreach value v in v's value list do
4. output entity pair (*Ep*, *Eq*) if sim(*Ep*, *Eq*) ≥ θ
5. end
6. end

Fig. 4: Functions for Load-balanced Matching

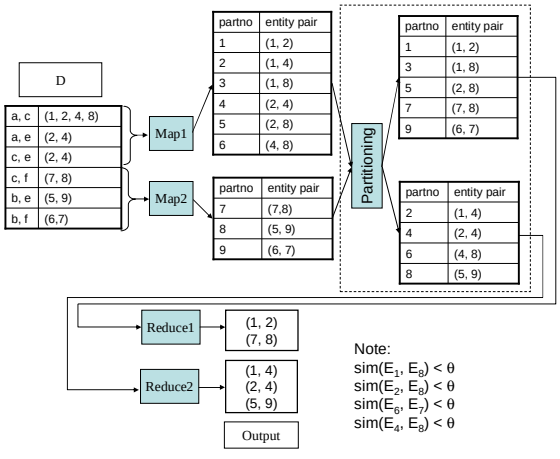


Fig. 5: An Example for Load-balanced Matching

5 Experimental Results

Extensive experiments were conducted to assess the performance of the proposed algorithm. Our experiments were performed in a 30-nodes cluster which contains three types of nodes, as shown in Table 1. All nodes run in Ubuntu 10.10, Hadoop 0.20.205.0, and Java 1.6.0.

Real dataset CiteSeerX (CiteSeerX 2012) was used in the experiments. CiteSeerX contains about 1.4 million publication records of total size 1.8 GB. Each publication record includes a record id, title, keywords, abstract, and URL information. The record id and its title were extracted and used as input because some records have no keywords. The stop-words in titles were removed and then the rest of the title words were used as keywords, which serve as the blocking keys. A record has 4.9 keys in average; the maximum number of keys is 20. Very few records contain only one key and they were skipped for the ER process since the minimum number of common keys *kc* is greater than one in our problem definition.

Config.	9 nodes	9 nodes	12 nodes
CPU	Intel Core i5	Intel Core i5	Intel E7600
	3.24 GHz * 4	3.24 GHz * 2	3.06 GHz * 2
Memory	3.4 GB		
Hard disk	SATA 500G		

Table 1: Experimental Environments

Same as the experiments in most studies on ER with MapReduce (Kolb et al. 2011a; Kolb et al. 2012a; Kolb and Rahm 2013), the execution time of comparing entities is not included in the result. That is, in the following context, the execution time accounts for the total time needed to find out the set of all the potentially similar entity pairs. In practice, after discovering these pairs, the complete ER process has to perform similarity computations like computing Jaccard coefficients for all these pairs. Clearly, a smaller number of the pairs results to a faster matching result. When the minimum number of common blocking keys (*kc*) is larger, the number of resulting pairs will be smaller. Nonetheless, a larger *kc* often indicates that the total number of combinations can be larger.

Table 2 shows that 94.3k combinations are generated and 53.8k resulting pairs need to be matched for *kc* = 2. There are 216.2k combinations are generated and only 36.5k resulting pairs need to be matched when *kc* is increased to 3. When *kc* is increased to 4, the number of combinations is the largest of 411.9k but only 19.5k pairs need further matching. When *kc* is increased to 5, the number of pairs required matching decreased to 8.4k.

Blocking keys	2	3	4	5
#combinations	94,365	216,230	411,962	209,366
#pairs	53,840	36,549	19,593	8,433

Table 2: Combinations and Results vs. Blocking Keys

Figure 6 shows the total execution time with respect to the number of common blocking keys (*kc*). The number of mappers (|M|) was 20, and that of reducers (|R|) was 13. The total execution time required for both combination-based blocking (legend *blocking*) and load-balanced matching (legend *load-balanced*) are displayed. Both times increases as *kc* increases. As mentioned above, the number of resulting pairs decreases rapidly as *kc* increases so that the real matching time decreases sharply in fact.

Table 3 indicates that the maximum load of the reducers decreases from 134MB to 56MB for *kc* = 5 after applying load-balanced matching. The data size to be handled before applying the proposed load-balanced matching is 2.3 times of that after applying the matching. In fact, the output data size after the combination-based blocking is 74MB for *kc* = 2, and up to 728MB for *kc* = 5. Without proper balancing of the reducers, many reducers would have to wait for the maximum-loaded reducer to complete its job. Take *kc* = 5 for example, before applying the balanced-matching, some reducers worked with nearly zero-sized data combinations and some were heavily loaded with 134MB data. The overall process cannot finish until this heavy-loaded reducer completes. Figure 7 depicts that the workloads can be effectively balanced so that the maximum load of the reducers can be decreased for all the settings of *kc*.

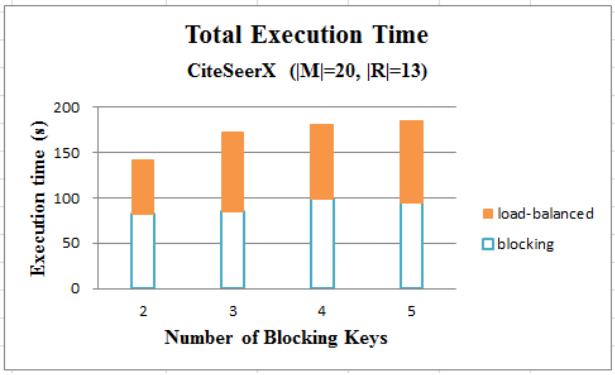


Fig. 6: Performance on Varying the Number of Blocking Keys

Table 3 indicates that the maximum load of the reducers decreases from 134MB to 56MB for *kc* = 5 after applying load-balanced matching. The data size to be handled before applying the proposed load-balanced matching is 2.3 times of that after applying the matching. In fact, the output data size after the combination-based blocking is 74MB for *kc* = 2, and up to 728MB for *kc* = 5. Without proper balancing of the reducers, many reducers would have to wait for the maximum-loaded reducer to complete its job. Take *kc* = 5 for example, before applying the balanced-matching, some reducers worked with nearly zero-sized data combinations and some were heavily loaded with 134MB data. The overall process cannot finish until this heavy-loaded reducer completes. Figure 7 depicts that the workloads can be effectively balanced so that the maximum load of the reducers can be decreased for all the settings of *kc*.

kc	2	3	4	5
Before	10.52M	82.97M	99.33M	134.09M
After	5.7M	11M	24.5M	56M

Table 3: Changes of Maximum Load of the Reducers

Next, the number of reducers was varied to evaluate the effects on total execution time. Figure 8 shows that the total execution time decreases as the number of reducers increases. As expected, more reducers may speed up the process. Note that the experiment uses an extreme large dataset by replicating CiteSeerX five times so that the differences of execution times can be exploited. When the size of input data is not very big, a MapReduce configuration with many reducers actually may cause extra-communications among the reducres. Thus, the dataset was enlarged to highlight the effects of the number of reducers.

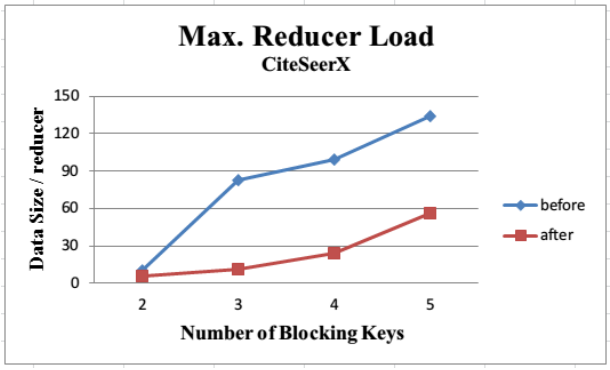


Fig. 7: Effects of Load-balanced Matching

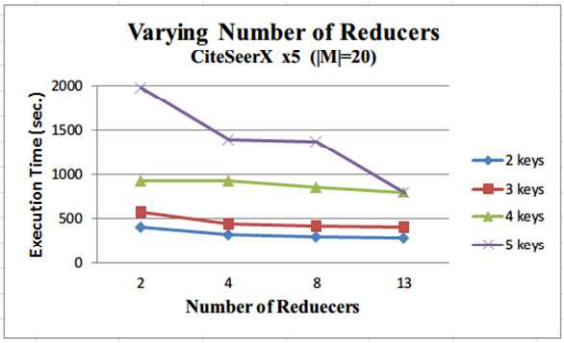


Fig. 8: Performance on Varying the Number of Reducers

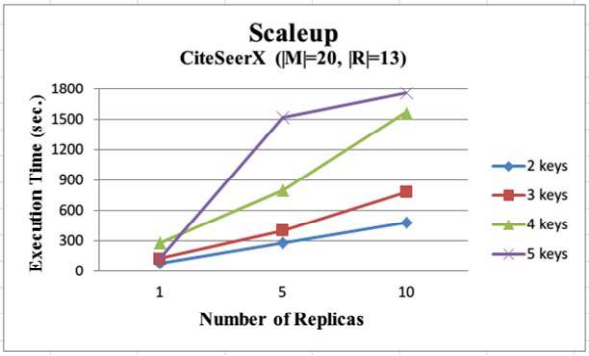


Fig. 9: Scalability of the Proposed Algorithm

The experimental result of evaluating the scalability of the proposed algorithm is shown in Fig. 9. The dataset CiteSeerX is replicated for 5 and 10 times, respectively. In Fig. 9, the total execution time increases linearly as the dataset size increases.

6 Conclusion

In this paper, we propose an algorithm to solve the entity resolution problem for big data analytics, using the MapReduce framework. The characteristic of multiple keys in entities is presented and utilized for effective key blocking (entity distribution) to improve the entity resolution process. The proposed algorithm features in the combination-based blocking and load-balanced matching. The combination-based blocking produces potential key combinations and distributes entity lists having common keys. The load-balanced matching generates entity-pairs combinations and balances the workloads of similarity computations among all the reducers in the MapReduce configuration. The experimental results show that the proposed algorithm may efficiently solve the entity resolution problem. Future works may extend this research to the entity resolution problem for the RS-join, such as the join between DBLP and the CiteSeerX.

7 Acknowledgement

The authors are grateful for the comments of the reviewers for improving the quality of the paper. This study is supported partly by the National Science Concil, Republic of China, under grant no. NSC102-2410-H-324-006.

8 References

Baraglia, R., De Francisci Morales, G., and Lucchese, C. (2010): Document similarity self-join with MapReduce. *Proc. IEEE International Conference on Data Mining*: 731-736.

Brizan, D. G. and Tansel A. U. (2006): A survey of entity resolution and record linkage methodologies. *Communications of the International Information Management Association* **6**(3): 41-50.

Chaudhuri, S., Ganti, V., and Kaushik, R. (2006): A primitive operator for similarity joins in data cleaning. *Proc. International Conference on Data Engineering*: 1-5.

Dean, J. and Ghemawat, S. (2004): MapReduce: simplified data processing on large clusters. *Proc. Symposium on Operating System Design and Implementation*: 137-150.

Elsayed, T., Lin, J., and Oard, D. W. (2008): Pairwise document similarity in large collections with MapReduce. *Proc. Association for Computational Linguistics on Human Language Technologies*: 265-268.

Hernández, M. A. and Stolfo, S. J. (1995): The merge/purge problem for large databases. *Proc. ACM SIGMOD International Conference on Management of data*, 127-138.

Kiefer, T., Volk, P. B., and Lehner, W. (2010): Pairwise element computation with MapReduce. *Proc. High Performance Distributed Computing*: 826-833.

Kim, Y. and Shim, K. (2012): Parallel top-k similarity join algorithms using MapReduce. *Proc. International Conference on Data Engineering*: 510-521.

Kolb, L. and Rahm, E. (2013): Parallel Entity Resolution with Dedoop. *Datenbank-Spektrum* **13**(1): 1-10.

Kolb, L., Thor, A., and Rahm, E. (2013): Don't match twice: redundancy-free similarity computation with MapReduce. *Proc. Workshop on Data Analytics in the Cloud*: 1-5.

Kolb, L., Thor, A., and Rahm, E. (2012a): Dedoop: efficient deduplication with Hadoop. *Proc. VLDB Endowment* **5**(12):1878–1881.

Kolb, L., Thor, A., and Rahm, E. (2012b): Load balancing for MapReduce-based entity resolution. *Proc. International Conference on Data Engineering*: 618-629.

Kolb, L., Thor, A., and Rahm, E. (2011a): Multi-pass sorted neighborhood blocking with MapReduce. *Computer Science - Research and Development* **27**(1): 45-63.

Kolb, L., Thor, A., and Rahm, E. (2011b): Parallel sorted neighborhood blocking with MapReduce. *Proc. Database Systems for Business, Technology, and Web*: 45-64.

Lin, J. (2009): Brute force and indexed approaches to pairwise document similarity comparisons with MapReduce. *Proc. ACM SIGIR Conference on Research and Development in Information Retrieval*: 155-162.

Lu, W., Shen, Y., Chen, S. and Ooi, B. C. (2012): Efficient processing of k nearest neighbor joins using MapReduce. *Proceedings VLDB Endowment* **5**(10): 1016-1027.

Metwally, A. and Faloutsos, C. (2012): V-SMART-Join: a scalable MapReduce framework for all-pair similarity joins of multisets and vectors. *Proc. VLDB Endowment* **5**(8): 704-715.

Pasula, H., Marthi, B., Milch, B., Russell, S. and Shpitser, I. (2002): Identity uncertainty and citation matching. *Proc. Neural Information Processing Systems*: 1401-1408.

Vernica, R., Carey, M. J., and Li, C. (2010): Efficient parallel set-similarity joins using MapReduce. *Proc. ACM International Conference on Management of Data*: 495-506.

Zhang, Q., Zhang, Y., Yu, H., and Huang, X. (2010): Efficient partial-duplicate detection based on sequence matching. *Proc. ACM SIGIR Conference on Research and Development in Information Retrieval*: 675-682.

CiteSeerX dataset URL.

http://asterix.ics.uci.edu/data/csx.raw.txt.gz (Accessed: 07/01/2012).

Hadoop URL.

http://hadoop.apache.org/ (Accessed: 08/24/2012).

Combining Pervasive Computing With Social Networking for a Student Environment

Elizabeth Papadopoulou[1], Sarah Gallacher[2], Nick K. Taylor[1], M. Howard Williams[1], Fraser R. Blackmun[1], Idris S. Ibrahim[1], Mei Yii Lim[1], Ioannis Mimtsoudis[1], Patrick Skillen[1] and Stuart Whyte[1]

[1]School of Maths and Computer Sciences,
Heriot-Watt University,
Riccarton, Edinburgh, EH14 4AS, UK
[2] Intel Collaborative Research Institute,
University College London,
London, UK

E.Papadopoulou@hw.ac.uk, s.gallacher@ucl.ac.uk, {N.K.Taylor, M.H.Williams, F.R.Blackmun, I.S.Ibrahim, M.Y.Lim, I.Mimtsoudis, P.Skillen, S.Whyte}@hw.ac.uk

Abstract

Whereas social networking has become an essential part of computing today, pervasive computing is seen as a key component for future systems. However, these two paradigms are complementary in many respects – the former responsible for communication and interaction between people, the latter focused on interaction with devices and services in the environment surrounding the user. By combining these two different paradigms in an integrated and seamless fashion one may provide users with the advantages of each plus the power obtained from using them together. Thus one might combine personalization, context awareness, learning, access to a wide range of devices and services, etc., with the management and operation of communities of users. This is the goal of the SOCIETIES project. By building on recent developments in pervasive systems and mobile computing, a new type of system that combines pervasive with social networking functionality – Pervasive Social Networking (PSN) – has been developed based on cloud and mobile technologies. Implementation of the basic system is complete and as part of the evaluation of the system it is currently being used by a group of students in a real user trial. This paper focuses on the student aspect and describes the requirements gathering exercises conducted with students. It then describes the architecture of the final system developed to meet the requirements. It ends with a brief outline of the final trial.

Keywords: Pervasive systems, social networking, mobile computing, cloud computing, smart spaces, ubiquitous systems.

1 Introduction

As the environment surrounding the user becomes more complex, with growing numbers of intelligent sensors and devices, so systems are becoming better able to change their behaviour to meet the user's needs. The goal of pervasive computing (Sun 2001) is to create an intelligent environment that provides support to the user in interacting with and managing these devices and services unobtrusively, without the user needing to be aware of and cope with the underlying communications and computing technologies. Driven by this important challenge, research in this area has followed a variety of different approaches with different objectives, and a growing number of prototypes have been created to test these. Examples include the Adaptive House (Mozer 2004), MavHome (Youngblood, Holder and Cook 2005), GAIA (Roman et al 2002, Ziebart et al 2005), Synapse (Si et al 2005), Mobilife (Strutterer et al 2007), Daidalos, Ubisec, etc.

On the other hand, social networking is a paradigm that has come into its own in a very short space of time. In recent years, online social networking has become one of the most significant trends in computer use, particularly through social network sites. In so doing it has significantly improved social connectivity between users and has opened up a whole new world of opportunities for exploiting the Internet. The unexpectedly rapid take-up of social networking services provided by systems such as Facebook, LinkedIn, MySpace, Bebo, YouTube, Flickr, etc., has transformed the way in which a large number of users use their systems, and takes up an increasing proportion of the time that the average user spends at his/her computer.

However, if these two different paradigms can be brought together and integrated seamlessly into a single system, there are significant benefits to be gained. The aim of the SOCIETIES project (Gallacher et al 2012) is to build on recent technical developments in these two areas to create such a system – a Pervasive Social Networking (PSN) system. This combines the strengths of pervasive systems with those of social networks to meet the needs of a wide range of different applications and users. Thus a PSN should enable the user to interact with devices in the environment, and communicate with other users either individually or as communities. Here a community is defined as a collection of participants who share some common characteristics or interests. In SOCIETIES a community may have its own criteria for membership, including the types of information that members are

prepared to share and third party services they may have access to.

In order to test out these ideas, the system is being evaluated in a set of field trials by three separate user groups. The three different user groups selected for this purpose are:

(1) Students. The motivation behind this choice is that students are very adaptive and take to new technology very easily. They also serve as an independent group of volunteers who are not employed by any of the developers of the system nor will they receive any academic credit for participating in the trials.

(2) Disaster Management. A set of professionals who meet together annually to simulate large scale disaster scenarios will assess the usefulness of the platform in disaster management situations.

(3) Enterprise. A collection of workers from industry have been evaluating the usefulness of the platform for handling support for delegates at a conference. This part has now been completed.

This paper is concerned with the student user group consisting of Computer Science and Information Systems students from Heriot-Watt University. Engagement with this group began in late 2010 when students were involved in the development of key scenarios. These were used to drive further design phases including the extraction of requirements and use cases. Since then the development of the platform has progressed and on 23rd October it was subjected to a full user trial. In this trial 20 students were provided with a device containing a prototype PSN platform to trial over a period of six weeks and their use of this monitored during this period.

The next section provides a brief background. The derivation of requirements through storyboarding and the immersive environment is described in section 3 while section 4 provides a brief introduction to the platform. Section 5 gives a detailed view of the components of the platform. Section 6 provides a brief description of the final trial and section 7 concludes.

2 Background

2.1 Pervasive Computing

The goal of pervasive computing is to create an intelligent environment in which devices provide unobtrusive connectivity and access to services, thereby improving user experience and quality of life without the user needing to be aware of and cope with the underlying communications and computing technologies. In this environment, the world around us is interconnected as a set of pervasive networks of intelligent devices that cooperate with each other and autonomously collect, process and exchange information, in accordance with the context and preferences of the user.

Pervasive computing embraces a wide range of diverse applications, including those of mobile computing systems and services. Driven by the important challenges presented by pervasive computing (Zaslavsky 2002), research in this area has followed a wide variety of different approaches with different objectives in mind, and a growing number of prototypes have been created to test different combinations of these. These diverse efforts can be categorized in various ways. For example, a brief summary of 29 software infrastructures and frameworks from a number of different projects is provided by Endres, Butz and MacWilliams (2005). This groups them into three main categories: Augmented Reality, Intelligent Environments and Distributed Mobile Systems.

Following a different way of grouping these developments, one major class of projects is that concerned with fixed smart spaces. A fixed smart space is a bounded physical environment filled with adaptive devices (such as lights, window shutters, etc.) that can be automatically managed to meet the needs of individual users. The main focus here lies in developing different forms of intelligent building, the most important of which is the Smart Home. This is motivated by the strong belief that pervasive technology can be used to provide safe and secure support for elderly and disabled citizens, which will facilitate their independent living and reduce the need for permanent carers or institutions. Besides covering the automatic control of devices providing lighting, temperature control, security, etc., the research has also extended to energy conservation in a smart building as well as a variety of intelligent appliances. Examples of systems of this type include the Adaptive House, MavHome, GAIA, Synapse, Ubisec (Groppe and Mueller 2005), etc.

Another major group has been that focused on mobile users with the aim of providing them with devices, networks and services to meet their needs wherever they may be. The location of the user plays a large part in the decision making process. The provision of support for mobile users and the problems associated with this, have been investigated in a number of research projects with corresponding prototypes developed to demonstrate and assess different approaches. For example, the European project Daidalos explored two separate architectures (Williams et al 2006), and developed prototypes based on each of these. By contrast, Mobilife focused on the issues of privacy and trust as well as on maintaining a “shared cognition” amongst groups of users. The project Spice (Cordier et al 2006) developed a platform for creating and executing mobile services.

The Persist project (Roussaki et al 2010) attempted to bridge the gap between these two classes of project by developing a prototype based on a Personal Smart Space (PSS) approach. This is a hybrid approach that can be used as a fixed smart space (taking advantage of sensor equipped buildings) as well as a mobile smart space that interacts with other surrounding fixed and mobile smart spaces (Papadopoulou et al 2010).

However, despite all the developments in this area, exploitation of these ideas has been slow.

2.2 Social Networking

In recent years, online social networking has become one of the most significant trends in computer use, particularly through social network sites. Using these sites, people can create accounts and connect digitally to friends, family, and others with ease. They can publicly share information and media about themselves and their lives, engage in chat with friends, form and join groups of users, and more. Certain sites, such as foursquare, add a location-based element to online social networking,

where users can check in to key locations and share their location details and histories with friends.

The popularity of social network sites is remarkable, and continues to grow. However, such sites are heavily geared towards networking in the digital realm. People have many real-world relationships that involve physical interaction to an extent that transferring them to a social network site is only somewhat useful. Even the aforementioned location-based services, while making some headway in bridging the physical-digital divide, are not exploiting the rich potential of real-world events and interactions to influence digital relationships and vice versa.

2.3 Combining the Two

The aim of the SOCIETIES project is to bring together the two different paradigms of pervasive computing and social networking to create a system that can benefit from both and from their combination.

In the case of pervasive computing the system enables the user to interact with devices in the vicinity. It monitors the user's actions and builds up a detailed history of these from which it can infer user preferences and user intent depending on the context of the user. These can then be used to assist the user by taking actions on the user's behalf when a relevant context is identified.

In the case of social networking the user controls the communications that he/she has with other users. This includes the information (text, pictures, etc.) that the user wishes to share, the other users who are allowed access to such information, and the access the user makes of other user's information. Communities (or groups) can be formed and within them subsets of information can be shared. And so on.

In combining the two the aim is to integrate the facilities of both. Thus the notion of monitoring user's actions and using learning techniques to identify patterns of behaviour, and hence user preferences and user intent, can be applied to personalising the user's interactions with social networking.

Moreover, a pervasive system can be in a very good position to detect apparent real-world relationships and communities, as well as the potential for new communities to be formed based on criteria such as shared interests, and to bring these into the digital realm. Thus it can be developed to identify potential new communities or existing ones that might be of interest to the user.

The use of context management within pervasive systems provides a rich source of context that can be exploited by a social networking system to provide a wider range of information with access controlled in a more context-dependent fashion. In turn the social networking systems can provide useful information on the user which can assist the pervasive system. By taking advantage of location information and social networking information on other users in the vicinity this can create opportunities for new applications in the future. However, all this must be done in the context of strict privacy controls to ensure the protection of user information.

3 Gathering Requirements

This section describes the requirements gathering exercises conducted with the students in preparation for the development of the SOCIETIES platform.

After a short introduction to the basic concepts of the SOCIETIES project, a combination of techniques (such as brainwriting, brainstorming and bodystorming) were performed with the students to identify scenarios that they felt were most useful or interesting for a PSN prototype to support. The feedback covered a wide range of situations from those that involved enhancing common practice to those that were novel. This exercise helped to identify the opportunity spaces for the PSN prototype in the everyday life of a group of student users.

This led to two preparatory user trials that took place in 2011. The primary objective of these trials was to record user response to an early low fidelity prototype of the proposed PSN system. The trials were carried out using two methodologies: *storyboarding* and *immersive environments*. Both techniques employ scenario based vision prototypes which serve the combined tasks of defining early design focus for developers and providing a site for evaluating user responses.

3.1 Storyboard Evaluation

The scenarios identified in the initial exercise were used to create a set of storyboard slides and an associated set of questions which were used to conduct a storyboard evaluation. Some fifteen first year Computer Science (CS) and Information Systems (IS) students took part in the session.

The storyboard slides detailed eight scenes that illustrated the SOCIETIES system supporting a student user in various situations. At key points during the slide presentations the students were asked multiple choice questions to gain their feedback on a concept that had just been presented. Each participant used a voting keypad to answer the multiple choice questions anonymously. The output from all keypads was captured on the session coordinator's laptop using voting system software.

A total of 19 questions were posed to the participants regarding the SOCIETIES concepts shown during the presentation of the storyboard slides. In this section, only the most significant questions and responses are presented in Table 1 although the entire result set is available by request from the project website (SOCIETIES project website).

#	Question	Response	Percentage
S1	*Would you have joined a "Freshers" community if this functionality had been available to you when you began University?*	**Yes**	**73%**
		Maybe	13%
		Only if the majority of other freshers had already joined	13%
		No	0%
S2	*Would you have found it useful to be told where other "Freshers" community members preferred to eat?*	**Yes**	**73%**
		Maybe	13%
		No, I'd rather discover such things myself	7%
		No, for other reasons	7%
S3	*Would you like to be automatically added to a community (without being asked for confirmation) related to your degree course?*	**Yes**	**73%**
		Maybe	13%
		No, I'd always like to be asked for confirmation first	7%
		No, for other reasons	7%
S4	*Would you have any privacy concerns about sharing your music preferences at a proactive disco?*	Yes	13%
		Maybe	13%
		No	**73%**
S5	*Would you like services to be automatically started on your behalf if the system was sure they would be of benefit to you?*	Yes	13%
		Maybe	13%
		No, I'd always like to be asked for confirmation first	**73%**
		No, for other reasons	0%
S6	*Would you appreciate suggestions to introduce yourself to new individuals (who share common goals/interests/characteristics)?*	**Yes, if we share something in common**	**53%**
		Yes, but only if we are significantly similar	27%
		No	20%
S7	*Do you think such technology would have improved your initial experiences at University in terms of making new friends?*	Yes	33%
		Maybe	27%
		No, I prefer to make friends the old fashioned way	**40%**
		No, for other reasons	0%
S8	*Would you join job/task sharing communities if they were available?*	**Yes**	**87%**
		Maybe	13%
		No	0%
S9	*Would you like your device to predict your behaviour and make suggestions to you?*	**Yes**	**47%**
		Maybe	27%
		No	27%
S10	*Would you mind being monitored if it meant that the system could make better suggestions to you?*	**No, as long as the system kept my data private and secure**	**60%**
		Maybe	33%
		Yes, even if the system kept my data private and secure	7%
S11	*Would you find automatic organising and planning of daily meetings/events useful in daily life?*	**Yes**	**67%**
		Maybe	27%
		No	7%

Table 1: Key results from the storyboard evaluation

3.2 Immersive Environment Evaluation

As with the storyboard evaluation, first year Computer Science and Information Systems students were invited to take part in the immersive environment evaluation. A total of thirteen students took up this invitation with each student being allocated a date and time for their individual immersive experience test which they attended alone. Each test took between ten and fifteen minutes to complete.

The immersive environment was erected in a test room and was designed to reflect physical locations that the students were familiar with such as University corridors and a meeting area. A number of devices were installed in the environment as interaction devices in accordance with the evaluation script. Figure 1 shows an aerial view of the immersive environment with hotspots and devices marked.

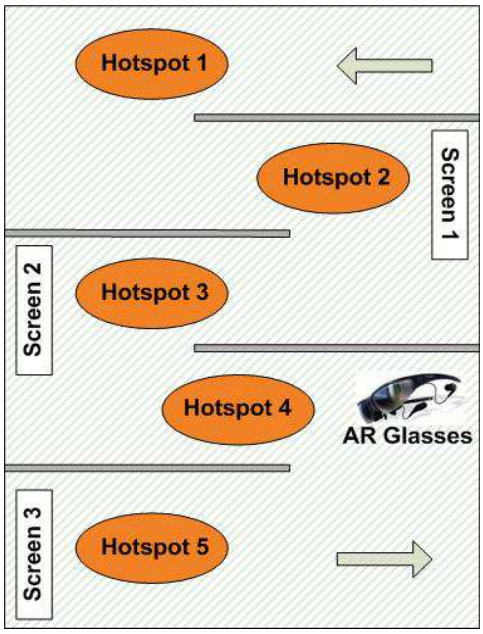


Figure 1: Aerial view of the immersive environment

Three screens acted as University advertisement screens that would show personalised content as the test participant walked past. The augmented reality (AR) glasses provided personalised content and other details in a more discrete fashion. An HTC smart phone acted as the participant's SOCIETIES device through which the trial participant could receive mock community alerts.

The controller was manipulated by the test coordinator to control and adapt the other devices within the environment appropriately. Several recording devices such as a camcorder and a dictaphone were also installed in the immersive environment to capture the reactions and feedback from participants. The immersive environment itself was a pathway through the various devices. The pathway was marked with five "Hotspots", each indicating a point where the participant would interact with a device or experience some SOCIETIES-like behaviour.

Each participant answered an average of 32 questions regarding the SOCIETIES concepts experienced within the immersive environment. The number of questions varied based on the decisions taken by participants during the immersive experience. In this section, only the most significant questions and responses are presented in Table 2 although the entire result set is available by request from the project website (SOCIETIES project website).

#	Question	Response	Percentage
I1	*Would you find information about the actions of other community members helpful to aid your own decision making?*	**Yes**	**100%**
		No	0%
I2	*Would you like it if screens around the University department displayed content specific to your interests when you walked past?*	**Yes**	**100%**
		No	0%
I3	*Are you ok with the University screens showing personal information (such as your name) in a public place?*	**Yes**	**62%**
		No	39%
I4	*Did you decide to join the community dedicated to one of your interests?*	**Yes**	**100%**
		No	0%
I5	*Would you still have joined the community related to your interests if you knew only 2% of your friends were members?*	**Yes**	**92%**
		No	8%
I6	*Would you find it useful to receive information (via AR glasses) on people you encounter in real life?*	**Yes**	**92%**
		No	8%
I7	*Did you decide to join the random "cardboard box artwork" community?*	Yes	31%
		No	**69%**
I8	*Would you find it useful to receive exam timetable information on the University screens?*	**Yes**	**92%**
		No	8%
I9	*Would you prefer the University screens to show course related info, personal info or both?*	Course-related	23%
		Personal-Interest	0%
		Both	**77%**
I10	*Did you decide to join the study group community related to one of your exams?*	**Yes**	**100%**
		No	0%
I11	*Would you still have joined the study group community if you knew only 2% of your friends were members?*	**Yes**	**85%**
		No	15%
I12	*If the study group community had requested your mobile phone number would you still have joined?*	**Yes**	**62%**
		No	39%
I13	*If the study group community had requested your location would you still have joined?*	**Yes**	**77%**
		No	23%
I14	*To minimise popups, would you allow the system to take decision on your behalf if it was 99% sure its decision*	**Yes**	**77%**
		No	23%

Table 2: Key results from the immersive environment evaluation

3.3 Comparison of Results

When the first trial (Storyboard Evaluation) was conducted, the aim was to present a set of imaginative scenarios to the participants and obtain their reactions to these. In doing so we were not inhibited by the constraints of actually demonstrating these scenarios. The second trial was much more focused and constrained by what we could do in the short amount of time that the participants were engaged in the trial. Although this had the disadvantage of being less imaginative, it had the advantage of letting the student actually experience the phenomena first hand.

Although it was not our intention to compare the results of the two trials, it was noticeable that, while both sets of results showed a general positive attitude towards SOCIETIES concepts, the results obtained from the immersive trial were in places more positive than those obtained from the storyboard trial.

In particular, this included:

(1) When queried about joining a community, in the storyboard trial participants indicated that this would depend on existing members whereas in the immersive trial they said it would not.

(2) In the storyboard trial participants did not like the idea of being automatically joined to any community whereas in the immersive trial there is evidence that automatic joining would be acceptable in certain cases.

(3) With regard to community information, in the storyboard trial most participants did not think community preferences would be useful whereas in the immersive trial all participants thought that this would be helpful.

(4) When asked whether they would like help in introducing them to other community members, most storyboard participants were unsure or against the idea whereas most immersive trial participants who used the AR glasses felt that this was really useful functionality.

(5) In the case of automatic behaviour, nearly all storyboard participants wanted to confirm before an automatic action was started whereas immersive trial participants were happy with some automatic actions on their behalf.

Thus, although it was not our intention to compare the two sets of results, especially since the number of participants is small, it did seem noticeable that, if participants actually experience the phenomena before being questioned about them, a slightly different result might be obtained compared to that obtained from a storyboard trial. In this case the results were more positive towards some of the concepts of a PSN platform.

4 SOCIETIES Platform

The requirements gathered from the three separate user groups were merged and an architecture derived that would provide the functionality needed to satisfy them. This architecture was based on the assumption that the main device with which the user interacts with the system is a smart phone. However, there may also be occasions when the user wishes to interact with the system via a laptop or PC. Whatever the case, since parts of the system require significant processing power, it has been assumed that the backend of the system resides in a cloud.

To simplify the architecture, it is divided into several layers, each of which incorporates various components and component blocks essential to the design of a PSN environment, as shown in Fig. 2.

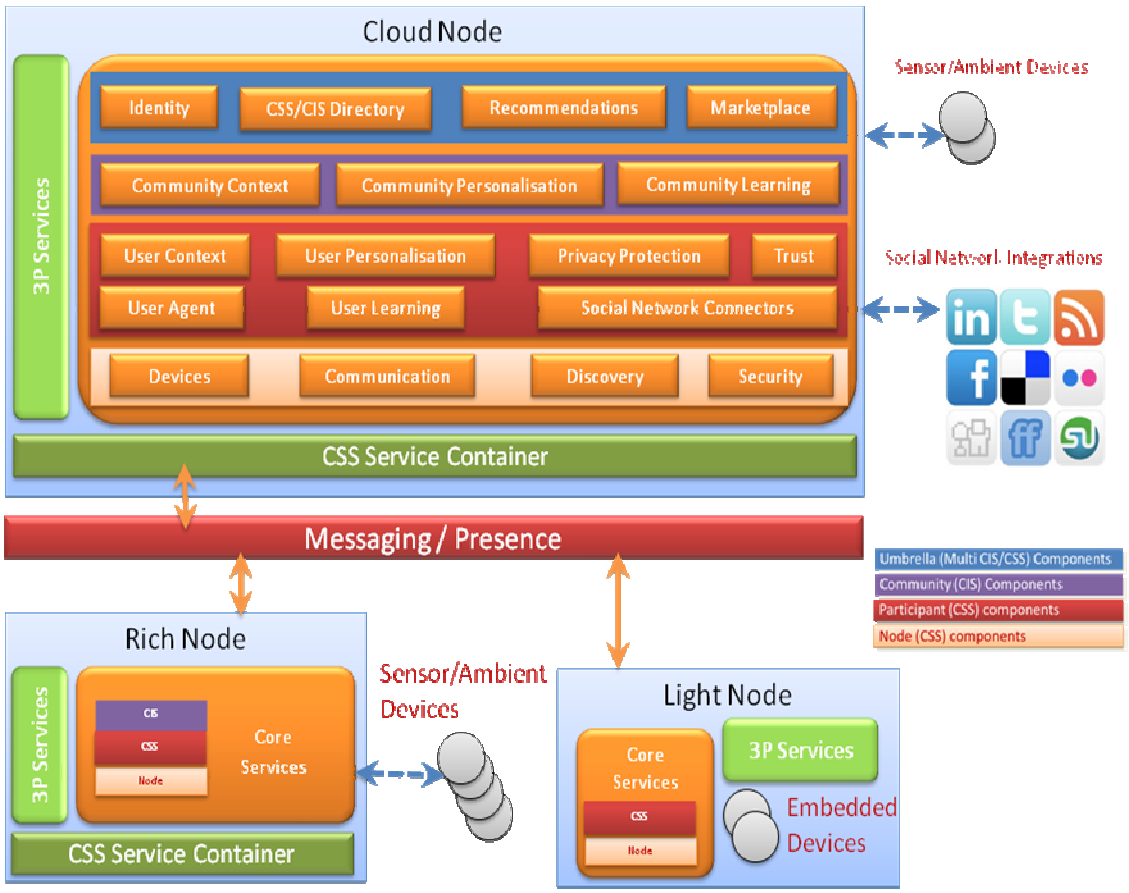


Figure 2: Architecture of the Societies Platform

Assuming that the main device with which the user interacts with the system, is a smart phone, we have based our implementation on an Android-based smart phone, although it is also possible to use other devices such as a notebook or laptop to interact with the system. Due to the more limited capabilities of a smart phone, the set of software components located on such a device is limited and provides minimum functionality. This is referred to as a Light Node. On the other hand a notebook or laptop may host more of the functionality of the system and is referred to as a Rich Node. In order to provide the full range of functionality on each type of node, both types communicate with a Cloud Node where most of the processing takes place.

Key to the system is the distinction between an individual user and a community of users. The different parts of the system that operate on behalf of a particular user are referred to as a Cooperating Smart Space or CSS. This represents a "smart space" of devices and applications that belong to that user. For example, the user may have both a smart phone and a laptop as well as several other smart devices. These together with the components in the cloud form a smart space for the user.

On the other hand, a community of users is referred to as a Community Interaction Space or CIS. When a community is formed, it is set up by a user through his/her CSS for a particular purpose. In general a community may have its own criteria for membership, including the types of information that members are prepared to share with each other. A community may also have associated with it a set of third party services that members may have access to.

5 Layers of the Architecture

The architecture can best be viewed as a layered one in which the Cloud Node contains the full set of components whereas the Light Node contains a minimal subset of them and relies on the Cloud Node to do most of its processing.

The Rich Node is somewhere between the two in that it has a more substantial subset of the components, making it possible to do more processing on the node without the constant dependence on the Cloud Node although it does still rely on the Cloud Node for some (e.g. offline data mining of the history data).

The four layers of the architecture are as follows.

5.1 Node Components

At the lowest level one has the node components themselves and the software needed for them to communicate with each other (Communication Framework) and to discover one another (Discovery).

At the very minimum the user will have a Light Node in the form of a smart phone and a Cloud Node. The Communication Framework provides the means for these two to communicate with each other. More generally the user may have other devices which can connect to the

Cloud Node and these too may use the Communication Framework.

In addition security is an important component at this level. This is mainly responsible for access control.

5.2 Participant Components

This layer contains the largest part of the system. It includes the main components providing functionality for the individual user or CSS. These include:

5.2.1 User Context

Context plays a key role in pervasive systems. Information about the context of a user is captured and stored in a Context Management system. Some of this information may be entered directly by the user (e.g. interests), other information is gathered from sensors or other devices and needs to be updated regularly.

Sometimes different sources may be used to provide information for the same attribute. Location is a good example. Out of doors one may use GPS to provide accurate location information while when the user is indoors one might use RFID tags to locate him/her.

In the SOCIETIES platform a Context Management system is used that keeps track of a range of different attributes, and uses three different methods to keep track of user location.

5.2.2 Personalisation

Personalisation is concerned with the set of techniques that are used to adapt the behaviour of the system to meet the needs and preferences of an individual user. Basically this means that under certain conditions (in certain contexts) the system needs to take specific actions on behalf of the user. These may involve setting parameters for a third party service, selecting or initiating a service, responding to a request for the user's personal information, etc.

In the SOCIETIES platform this subsystem uses two very different approaches to determine when to take action and what action to take. The first is based on user preferences. These can be viewed as rules of the form:

IF a context arises THEN perform some action

although in practice the process is more complex.

The second is referred to as User Intent and is based on sequences of actions that are performed by the user in particular contexts. Thus if the system detects that the user is part way through a known action sequence, it can predict what action to perform in the future provided a suitable context match arises.

In the SOCIETIES platform two different techniques are used for each of these two different approaches.

5.2.3 Learning

To build up a set of user preferences, one cannot expect the user to provide these manually. Instead the system monitors the user's actions plus the context in which they occur and uses this to "learn" the user's preferences.

Since two different techniques are used for handling user preferences in the SOCIETIES platform, two different styles of learning are required. The first technique used is based on a neural network and learning for this is straightforward. The second technique is based on preference rules and a variation of the C4.5 algorithm is used. This is coupled with a confidence level indicator which provides a measure of the degree of confidence associated with a preference rule at any stage.

In addition to these the project is also experimenting with the use of a Bayesian Network to handle input from bio-sensors. Again learning is straightforward.

5.2.4 User Agent

With all these different techniques being used to predict actions for the system to perform on the user's behalf, an arbiter is required to select the most appropriate one. The User Agent is the component responsible for taking the outputs from these different techniques and deciding which to perform.

It is also responsible for communication with the user. Thus whenever the system decides to perform an action on the user's behalf, the User Agent informs the user and provides the user with an opportunity to reject this if the action is not what he/she wants. If the user does nothing, the system proceeds with the action.

5.2.5 Privacy

Protection of user privacy is essential in a system where the user's personal information may be shared with other users. In the SOCIETIES platform the Privacy component is responsible for providing the support to enable the user to manage personal information and its disclosure.

To determine what data attributes may be disclosed and to whom, the system uses the process of Privacy Policy Negotiation between the preferences of the user and the requests for data from third party services or communities. To determine in what form the data should be released, a process of obfuscation is employed. And to provide further protection to the user a system of multiple identities is used.

5.2.6 Trust

The decision to share information with another user or a third party service does rely to some extent on the degree of trust that the user has in the other user or the third party service.

As the number of contacts a user has and the number of third party services available to a user increases so the need for the user to assess the trustworthiness of these entities becomes increasingly important. The set of communities to which a user belongs can be used to provide support for the trust assessment mechanism.

5.2.7 Social Network Connectors

By enabling the system to connect to social network sites directly though the interfaces provided by the social network systems, the SOCIETIES platform can access information about users directly and provide this to the components of the system that might use it. In particular the communities within SOCIETIES can benefit from information on their members obtained from these sites, as can third party services.

In the current state of the system one can obtain information from social network sites but not write information to them.

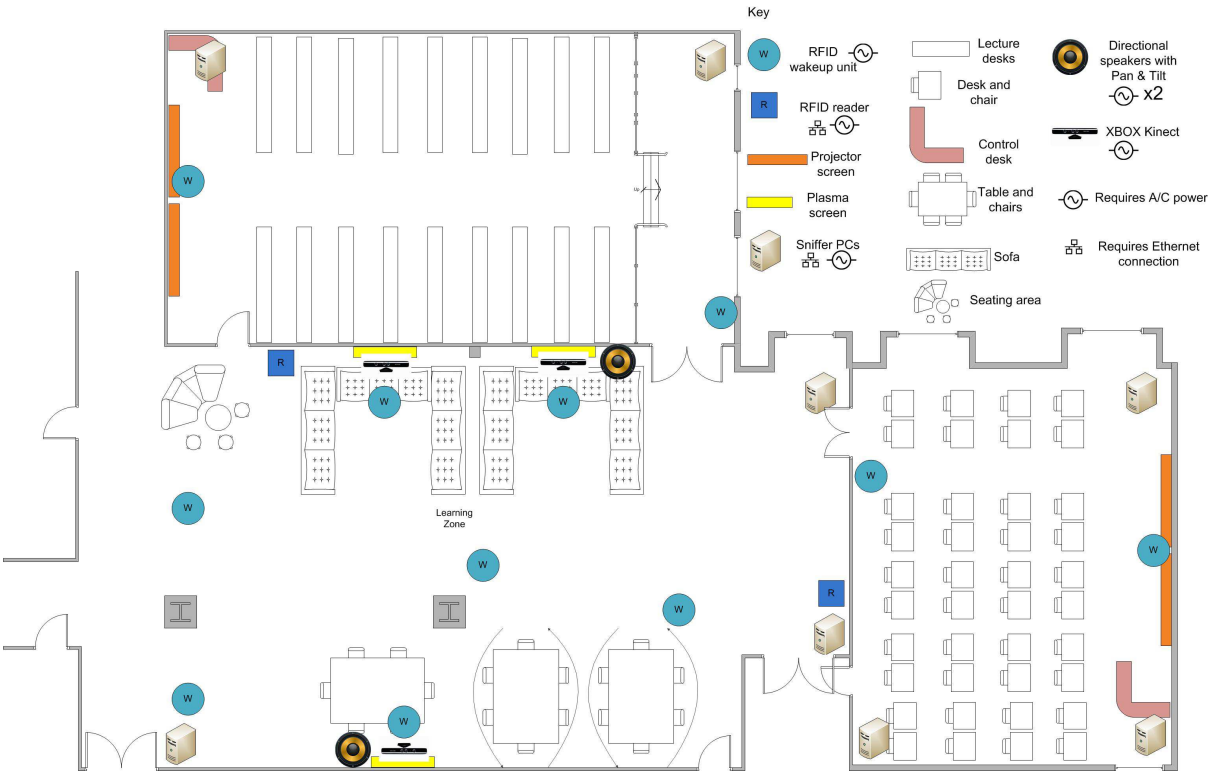


Figure 3: Set up of pervasive Learning Zone

5.3 Community Components

The Community Component Layer contains three components that provide functionality relating directly to communities. These are as follows.

5.3.1 Community Context

Just as the individual user has an associated set of context attributes pertaining to that user, each community may have context attributes associated with it that are derived from the attributes of its individual members.

An important example of a community context attribute is location. When a number of members of a community are gathered together, it may be useful for other members to know where this is taking place. Other community context attributes may be derived from mean or median values of the attributes of its members (e.g. average age).

5.3.2 Community Personalisation

Here the notion of a user preference has been extended to that of a community preference. This takes the same form as an individual user preference and hence can be used in the process of personalisation in the same way.

5.3.3 Community Learning

The SOCIETIES platform provides a mechanism for processing the individual user preferences of the members of a community to infer or "learn" the common preferences associated with the community. Each individual member can choose to inherit some or all of these. This is particularly important for new members joining a community.

5.4 Umbrella Components

This layer lies outside the other three layers and provides functionality which applies to all CSSs/CISs. There are four components in this layer. These are:

(1) CSS/CIS Directory – which provides typical directory services;

(2) Identity – which controls the unique identities allocated to CSSs;

(3) Recommendations – which recommends relevant existing or potential CISs to CSSs;

(4) Marketplace – which provides access to third party services for users.

6 Full User Trial

The full user trial started on 23rd October and is scheduled to run for six weeks. A group of 20 student volunteers have been issued with RFID tags and smart phones (Samsung Galaxy SIII) loaded with the software for Light Nodes.

The main area where the system is being used is the Learning Zone in the School of Mathematical and Computer Sciences. This is an area adjacent to two main lecture rooms that is furnished with tables and chairs used by students for work and relaxation. This has been equipped with RFID wakeup units and readers, large plasma screens, XBOX Kinects, etc. as shown in Fig. 3. In addition we have two servers hosting the Cloud Nodes which contain the basic system as well as nine third party services.

7 Summary and Conclusion

Pervasive computing and social networking are two complementary paradigms which the SOCIETIES project aims to bring together to create a Pervasive Social Networking System. A system has been created involving a combination of mobile devices (smart phones and laptops), cloud computing and devices in the user's environment to provide the basic functionality required. In addition a number of third party services have been developed to run on this system. The system is currently being evaluated by three trial groups. This paper focuses on one of these groups, namely university students.

At the outset of the SOCIETIES project, a student demographic was identified, and this group has been involved in several activities, with the intention of capturing their requirements for such a system. Through this process they have also been exposed to ideas of pervasive computing and what the SOCIETIES system can offer them. By the time that the full trial began on 23rd October we believe that they had sufficient understanding to be able to make full use of the system.

Section 3 describes two exercises conducted with the students to establish requirements for the system. The first was a storyboard evaluation, the second an experiment with an immersive environment.

From the requirements from all three user trial groups the design for a PSN was derived and the architecture of this is described in section 4 with details of the components given in section 5.

8 Acknowledgements

This work is supported by the European Union under the FP7 programme (Societies project) which the authors gratefully acknowledge. The authors also wish to thank all colleagues in the Societies project. However, it should be noted that this paper expresses the authors' personal views, which are not necessarily those of the Societies consortium. Apart from funding the Societies project, the European Commission has no responsibility for the content of this paper.

9 References

Cordier,C., Carrez, F., Van Kranenburg, H., Licciardi, C., Van der Meer, J., Spedalieri, A., Le Rouzic, J.P. and Zoric, J. (2006): Addressing the Challenges of Beyond 3G Service Delivery: the SPICE Service Platform. *Proc. Workshop on Applications and Services in Wireless Networks (ASWN '06)*.

Endres, C., Butz, A., and MacWilliams, A. (2005): A survey of Software Infrastructures and Frameworks for Ubiquitous Computing, *Mobile Information Systems Journal* **1**:41-80.

Gallacher, S., Papadopoulou, E., Taylor, N.K., Blackmun, F.R. and Williams, M.H. (2012): Intelligent Systems that Combine Pervasive Computing and Social Networking. *Proc. Ninth International Conference on Ubiquitous Intelligence and Computing (IEEE UIC 2012)*, Fukuoka, Japan, 151-158, IEEE Computer Society.

Groppe, J. and Mueller,W. (2005): Profile Management Technology for Smart Customizations in Private Home Applications. *Proc 16th International Workshop on Database and Expert Systems Applications (DEXA'05)*, 226-230.

Mozer, M.C. (2004): Lessons from an Adaptive House. In *Smart Environments: Technologies, protocols and applications.* Cook, D. and Das, R. Eds. 273-294.

Papadopoulou, E., Gallacher, S., Taylor, N. K. and Williams, M. H. (2010): Personal Smart Spaces as a Basis for Identifying Users in Pervasive Systems. *Proc. International Workshop on Ubiquitous Service Systems and Technologies (USST 2010)*, Xian, China, 88–93, IEEE CS Press.

Roman, M., Hess, C.K., Cerqueira, R., Ranganathan, A., Campbell, R.H. and Nahrstedt, K. (2002): Gaia: A middleware infrastructure to enable smart spaces, *IEEE Pervasive Computing* **1:**74-83.

Roussaki, I.. Kalatzis, N., Doolin, K., Taylor N. K., Spadotto, G., Liampotis, N. and Williams, H. (2010): Self-Improving Personal Smart Spaces for Pervasive Service Provision. In *Towards the Future Internet*, Tselentis, G., Galis, A., Gavras, A., Krco, S., Lotz, V., Simperl, E., Stiller, B. and Zahariadis, T., 193-203, IOS Press.

Si, H., Kawahara, Y., Morikawa, H. and Aoyama, T. (2005): A stochastic approach for creating context aware services based on context histories in smart Home. *Proc. 1st International Workshop on Exploiting Context Histories in Smart Environments, 3rd International Conf on Pervasive Computing (Pervasive 2005)*, 37-41.

SOCIETIES project website, http://www.ict-societies.eu

Strutterer, M., Coutand, O., Droegehorn, O. and David,K. (2007): Managing and Delivering Context-Dependent User Preferences in Ubiquitous Computing Environments. *Proc. International Symposium on Applications and the Internet Workshops (SAINTW '07)*.

Sun, J. (2001): Mobile ad hoc networking: an essential technology for pervasive computing. *Proc. International Conference on Info-tech & Info-net*, 316-321.

Williams, M. H., Taylor, N. K., Roussaki, I., Robertson, P., Farshchian, B. and Doolin, K. (2006): Developing a Pervasive System for a Mobile Environment. *Proc. eChallenges 2006 – Exploiting the Knowledge Economy*, 1695 – 1702.

Youngblood, M.G., Holder, L.B. and Cook, D.J. (2005): Managing Adaptive Versatile Environments. *Proc. 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom '05)*, 351-360.

Zaslavsky, A. (2002): Adaptability and Interfaces: key to efficient pervasive computing. *Proc. NSF Workshop on Context-Aware Mobile Database Management*, 24-25.

Ziebart, B. D., Roth, D., Campbell, R. H. and Dey,A. K. (2005): Learning Automation Policies for Pervasive Computing Environments. *Proc. 2nd International Conference on Autonomic Computing (ICAC '05)*, 193-203.

Developmental Directions in Parallel Accelerators

K.A. Hawick **D.P. Playne**

Computer Science, Institute of Natural and Mathematical Sciences
Massey University – Albany
North Shore 102-904, Auckland, New Zealand
Email: {k.a.hawick, d.p.playne}@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

Abstract

Parallel accelerators such as massively-cored graphical processing units or many-cored co-processors such as the Xeon Phi are becoming widespread and affordable on many systems including blade servers and even desktops. The use of a single such accelerator is now quite common for many applications, but the use of multiple devices and hybrid combinations is still very unusual. The main barrier to greater uptake of multiple accelerators in applications is still the software ecosystem and in particular the interoperability limitations of setting up appropriate software stacks for novel accelerator combinations. We present some benchmark results for various multiple and hybrid accelerator combinations using some up to date modern devices and discuss feasible developmental directions for high computational performance scientific applications software to use them. We compare results with equivalent benchmarks on conventional multi-cored CPUs.

Keywords: accelerator; GPU; Xeon phi; massive core; multi core; scientific applications; software ecosystem.

1 Introduction

An ability to exploit concurrency(Oskin 2008) in user application programs(Sutter & Larus 2005, Cantrill 2006) and not just at the operating system level(Knauerhase, Cledat & Teller 2012, Kleidermacher 2008) is increasingly important. Most users will typically have at least two and possibly four cores in their desktop CPU at the time of writing. Six cores per CPU is already common and eight core Intel CPUs(Reinders 2007) or 16-core AMD CPUs(Butler, Barnes, Sarma & Gelinas 2011) will likely become common.

Processing accelerators continue to be a widely used solution to attaining good computational performance while avoiding excessive heat density issues from designing CPUs with overly high clock speeds. There remain a number of different approaches to providing parallelism in such accelerators however. These are typified by the many fine grained cores found in Graphical Processing Units (GPUs) from vendors such as NVidia, and the many integrated traditional CPU cores used in Intel's MIC/Xeon Phi devices.

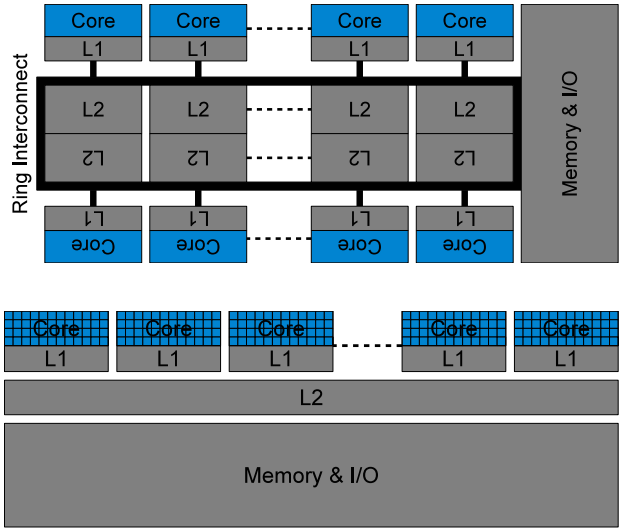


Figure 1: Intel Xeon Phi (top) and NVidia Kepler GPU (bottom) accelerator architectures.

These vendors have very different approaches to servicing their processors in terms of cache, memory and bus structures(Patterson & Hennessy 2009) as well as managing potential concurrency through the varying accelerator architectures.

Figure 1 shows a summary of the architectures of the Intel Xeon Phi with its ring based bus connecting around 60 conventional CPU cores (above) and the multi streaming architectural approach of NVidia's Tesla GPU accelerator with current models having around 2000 cores.

Graphical Processing Units (GPUs) and other devices(Leist, Playne & Hawick 2009) have already proven themselves a valuable mechanism to speed up many applications using a data-parallel programming model that can accelerate a conventional CPU. It is even becoming economically viable to host multiple GPUs on a single CPU host node(Hawick & Playne 2013), and clusters using this multi-GPU assisted node model are becoming prevalent.

We are exploring a range of combinations of multi-core CPUs, running various thread management software systems to manage their multiple GPU accelerators. We anticipate significant flexibility and scalability achievable using this approach and believe it has major implications for future generation HPC systems including clusters and supercomputer facilities.

Many of the supercomputers in the present Top 500 list(TOP500.org n.d.) now incorporate graph-

ical processing unit devices as accelerators for the individual nodes. Such systems use fast interconnection technologies such as InfiniBand or a proprietary technology instead of regular Ethernet switching technology. These interconnects aim at providing fast and low latency interconnections between individual nodes, each of which will typically run a full operating system software stack. Applications programmers will typically use a software model such as message passing - implemented with for example MPI or a variant - to communicate between participating nodes in a single MIMD program.

Individual nodes however can be accelerated by one or more devices. Although other accelerators such as customised field programmable gate arrays are sometimes used, at the time of writing GPUs are becoming a very popular acceleration devices. Typically a hosting CPU node is accelerated by one or more GPU devices which are connected to the CPU using PCI or PCI Express bus technology.

Applications programmers invoke special codes on the GPU that are called from the normal CPU program. The GPU typically offers a very large number of simpler cores than those found on a CPU. The parallelism model most likely used on GPUs is that of fine grained data parallelism whereby very many threads are used to manage data-parallel operations. Modern GPUs often are structured with some shared facilities such as floating-point units(FPUs) – not all the individual cores will have an FPU and they are typically grouped together so that for example some sort of multi-processing unit with floating point capability acts to group together individual simpler cores.

The CPU hosting program can in fact be multiple threaded, and this approach is often used to make use of the multiple cores that a modern CPU will typically incorporate. It is also possible for these cores to run more than one thread and this approach is often favoured if there are slow memory or data access operations required. A single hardware CPU core will quite often be given two or more applications threads to run to allow data accesses to be suitably interleaved with computations. Modern CPUs will often have special extra hardware capabilities to explicitly support this approach.

We therefore find that there are several hierarchical layers of parallelism available to exploit on a multi-noded, multi-cored, GPU-accelerated supercomputer system. Different applications and user schedules can make use of these in many different manners to exploit the overall hardware resource to best effect. The following terminology is useful to describe this hierarchy of levels of parallelism, where we give some references to our recent work against each relevant category:

Grid: Distributed and often separately owned supercomputing resources can be used

Node: Individual Nodes connected with InfiniBand or Ethernet switching technology

Multi-CPU: Multiple CPUs on multi-socket motherboard share node main memory

CPU: Individual CPU has multiple cores of high individual capability including their own floating point unit

CPU-Core: CPU core can run more than one application thread

Accelerator-Device CPU thread can be accelerated by an additional device such as a GPU or a MIC

GPU-MP: GPU device has some number of individual multi processors, usually with their own floating point unit and controlling some group of lower-level simple cores

Accelerator Core: Individual fine-grained vector or SIMD operations carried out by the accelerator core

There is potential scope for work at all levels of this hierarchy. Practicably, the financial resources required to work at the high end of supercomputing are now restricted to a very few institutions. Nevertheless it is quite feasible to progress systems software, library development, applications development and other parallel computing software research anywhere. In this present article we focus on the use of accelerator devices and how they are programmed and perform.

Our article is structured as follows: In Section 2 we summarise some of the important details for the devices we consider in this article, including Intel Xeon CPUs and AMD Bulldozer CPUs and Intel Xeon Phi MIC and NVidia Kepler GPU accelerators. We describe some benchmark codes we have used to compare the attainable performance on these platforms in Section 3 and discuss the various parallel and threading software approaches that can be used to program them in Section 4. We give some performance results in Section 5 and a discussion of developmental directions for accelerators in Section 6 and offer some conclusions and areas for further work in Section 7.

2 CPUs and Accelerators

In this article we consider four principle devices: the Intel Xeon CPU; the AMD Bulldozer CPU; the Intel Xeon Phi many-cored accelerator; and the NVidia Kepler GPU accelerator - all of which represent the state of the art at the time of writing. The CPUs have 2,4,6,8, or 16 cores and come as conventional socket mounted chips on the motherboard - albeit with substantive heatsinks cladding them. The accelerators both come as PCI bus slot mounted cards usually will built in separate cooling units. While it is notoriously difficult to make sense of price performance data especially given currency exchange spot rates, as a historical record, the devices we consider cost approximately NZ$1,000 - NZ$2000 for the CPUs, and around NZ$4,000 - NZ$6,000 for the accelerator cards at the time of writing.

2.1 Intel Xeon

The Intel Xeon processors come in many different models and architectures but all follow the same principle design of a multi-core processor. Currently 2 to 8 processing cores, each of which is capable of supporting two threads with Intel® Hyper-Threading. Each of these cores accesses memory through a number of caches to improve memory access performance (Intel 2013). The main Xeon processor used for evaluation in this work is the 8-core Xeon E5-2687W at 3.10 GHz.

2.2 AMD Bulldozer

The Bulldozer architecture from Advanced Micro Devices (AMD) is a multi-core CPU design split into modules. Each of these modules consists of two cores

capable of executing a single-thread each. These Bulldozer cores also have levels of cache to improve memory performance (Butler et al. 2011). The Bulldozer CPU tested in this paper is the AMD Opteron 6274 running at 2.2 GHz.

2.3 Intel Xeon Phi

Our Xeon Phi cards have 8 memory control units and 60 cores running at 1.05 GHz and connected in a ring network. Each core has 512kB L2 cache and is essentially a 64-bit "Knights Corner" architecture capable of supporting 4 hardware contexts at once and using Intel's thread apparatus, needs at least two running threads to make best use of the clock cycles(Saule, Kaya & Catalyurek 2013).

2.4 Nvidia Kepler

The Kepler architecture is the current generation GPU design from Nvidia. The GK110 architecture consists of 15 SMX processor units with six memory controllers (NVIDIA 2012). Each of these SMX units contains 192 cores and various optimised memory areas. The most significant change from the early GPU designs (Leist et al. 2009) has been the introduction of L1 and L2 cache into the memory subsystem.

3 Benchmarks

A challenge in benchmarking processing devices with very different architectures is choosing a suitable benchmark task or test that can be expressed portably in a manner that can actually be encoded on all devices under consideration fairly. There have been many good historical benchmark suites developed including the Livermore Loops(McMahon 1986) which are mostly simple computational tasks expressed as iterative loops and the now well known Lapack linear algebra benchmarks(Demmel, Dongarra, DuCroz, GreenBaum, Hammarling & Sorensen 1988, Bischof & Dongarra 1989).

Porting even these well known benchmark codes to new platforms represents quiet a lot of work and in many cases it is not always even fair to compare disparate platforms with application benchmarks that may represent quite particular areas of optimisation speciality.

For the work reported here we have chosen three very simple tasks that are reminiscent of some of the Livermore loop cases but which can be expressed as full code explicitly here. These are based on the notion of generating and using pseudo random numbers as part of a Monte Carlo or other stochastic calculation or simulation.

```
double sum = 0.0;
for(int i=0;i<N;i++) sum += rng->uniform();
```

Figure 2: Simple benchmark timing generation of summed random numbers.

The code listed in Figure 2 shows the first of these three simple codes. It simply generates some number N uniform random numbers, and sums them. Explicit summation is needed to ensure an overly smart optimising compiler does not optimise out the contents of the loop. We can make the random number generator as high quality as we like and can concern ourselves with details such as whether it is 32-bit, or 64-bit or which algorithm is used. The generator algorithm that is often used in quality Monte Carlo work at the time of writing is the Mersenne-Twistor algorithm(Matsumoto & Nishimura 1998). However the RAN4 generator from the Numerical Recipes suite (Press, Teukolsky, Vetterling & Flannery 2007)is just as good for our purposes and is also easily implemented. Other studies have considered random number generation for accelerators in more detail(Giles 2009), but the important notion for us is simply that the algorithm be portable and implementable on all platforms under consideration.

Our first code will give a measure of how well each processing device performs at raw computation - albeit mixing floating point and integer arithmetic operations as would a typical actual application code such as a simulation.

```
double sum = 0.0;
double *list; list = new double[N];
for(int i=0;i<N;i++) list[i]=rng->uniform();
for(int i=0;i<N;i++) sum += list[i];
```

Figure 3: Stored case.

The code listed in Figure 3 shows how the system's memory management system interacts with computation which still allowing vector and pipelining optimisations to be introduced and exploited by the optimising compiler. The randomly generate d numbers are stored in a large vector in order and then summed in order in a separate loop. The memory access predictability should allow a compiler to exploit cache management and pre-fetching and other associated memory management optimisations. This pattern is also fairly typical of a regular structured calculations found in a simulation application.

```
double sum = 0.0;
int *order; order = new int[N];
double *list; list = new double[N];
for(int i=0;i<N;i++) order[i] = i;
for(int i=0;i<N;i++){
  int j = rng->int64() % N;
  int k = rng->int64() % N;
  int swap = order[j];
  order[j] = order[k];
  order[k] = swap;
}
for(int i=0;i<N;i++) list[i] =  rng->uniform();
for(int i=0;i<N;i++) sum += list[order[i]];
```

Figure 4: Indirectly addressed and shuffled stored case.

The code listed in Figure 4 represents a more challenging set of tasks for devices. In this case the random numbers are stored in the same linear list as in case 2, but this time they are summed in a deliberately random order. This should deliberately confound cache and memory management predictions and pipelining and in some sense ought to represent a worst case limit on the platform's ability to do memory-bound computations.

We need to chose the size N of these tasks with care. In practice one finds that modern desktop and blade systems at the time of writing will have clock speeds of the order of a few GHz and affordable memory sizes of the order of at least a few GBytes - and in some cases a few tens of GBytes. A size N of approximately $10^8 - 10^9$ ought to be practicable in terms of tasks that can fit in memory and which take at least of the order of seconds to complete.

A great deal has been reported in the literature concerning the proper use of clocks and clock resolutions for computer benchmarking(Bailey, Barscz, Barton, Browning, Carter, Dagum, Fatoohi, Fineberg, Frederickson, Lasinski, Schreiber, Simon, Venkatakrishnan & Weeratunga 1994, Grove & Coddington 2005). For our purposes here we want to ensure the times taken are always sufficiently large compared to clock granularities or uncertainties that we can make fair comparisons between the different platforms and draw fair conclusions therefrom.

While it is useful to consider absolute times to take account of the different speed/price advantages of individual systems, speedup measured in the classical manner is useful to compare the scalability of the multi-threaded software benchmarks on the mix of systems being compared.

Parallel speedup for n processors - or threads in this present work - is usually defined as:

$$S_n = \frac{T_1}{T_n} \tag{1}$$

where T_1 is the time taken for a single thread and T_n the time taken for n such threads. It is important to estimate the experimental uncertainty in this, and we obtain this from experimental standard deviations made in a set of times. Although a computer ought to be a completely deterministic device, there are a number of factors that can lead to a spread of measurements.

Any modern computer system typically has a large number of independent processes running to service the operating system. While one can minimise the number of unusual interrupts or events by temporarily isolating computers from the network and limiting user access for duration of benchmarking, inevitably there are operating systems processes that wake up and consume resources briefly. Generally experience shows that these will contribute only a small uncertainty to benchmark times providing suitable averaging is done over for example 10 or more independent runs.

Common sense indicates that clock accuracies are limited and any timing measurement that is comparable in size to CPU clock granularities will inevitably be contaminated by context switching times. While it is possible on some systems it is possible to obtain very accurate clock timings, it is advantageous for this sort of study to be able to use the portable and readily available systems clock. This is not necessarily more reliable than to a resolution of around milliseconds. The benchmarks have therefore been tuned so that the repeated number of operations performed lead to times of the order of tens of seconds or higher. These should therefore dominate any millisecond fluctuations or indeed any process or OS daemon context switching times of sub second scale. This means that the measured times should be representative - on average - of the quantities of actual interest and that conclusions can be safely drawn.

Using the usual first order calculus of derivatives to study the propagation of uncertainties in 1, we find that the uncertainty in speedup δS_n is obtained by adding the **relative** uncertainties in the times T_1 and T_n.

$$\delta S_n \approx \left(\frac{\delta T_1}{T_1} + \frac{\delta T_n}{T_n} \right) S_n \tag{2}$$

For our purposes here we simply use the standard deviations in the distribution of measured values as the uncertainty values for δT_1 and δT_n.

This means that all our data points are heavily dependent upon a reliable and accurate measure of T_1 - the time for a single thread. In the results that follow, we therefore average over around 10 individual timing measurements to minimize this uncertainties and which we plot as error bars on the diagrams.

4 Parallel Software

The results gathered on the Intel Xeon Phi are still preliminary results and only explore one of the methods of parallel execution on this co-processor. The benchmark tests executed on this device have been implemented using Intel Threading Building Blocks (TBB)(Intel 2010, Reinders 2007) and executed in native execution mode on the Xeon Phi. Other parallel implementation options for the Xeon Phi include pThreads (IEEE 1995), Intel Cilk threads (Blumofe, Joerg, Kuszmaul, Leiserson, Randall & Zhou 1995) and OpenMP (Chandra, Dagum, Kohr, Menon, Maydan & McDonald 2001).

The native execution mode involves copying the executable directly to the device. The other option is the offload mode where a program executing on the host CPU can offload execution onto the Xeon Phi, however this can include additional overheads for copying data. All code for the Xeon Phi has been compiled using the Intel Compiler and is running on Centos 6.4. A code listing showing the simple test case implemented using TBB can be seen in Listing 1, this code can be executed directly on the Xeon Phi.

Listing 1: Intel Threading Building Blocks Implementation of the simple test case.

```
double simple() {
  double total = 0.0;
  parallel_for(blocked_range<unsigned long>(0, N),
      [](const blocked_range<unsigned long> &r) {
    RNG rng;
    double subtotal = 0;
    unsigned long i;
    for (i = r.begin(); i!=r.end(); ++i) {
      subtotal += rng.uniform();
    };
    tbb::spin_mutex::scoped_lock lock(add_mutex);
    total += subtotal;
  });
  return total;
}
```

The NVidia GPU benchmarks have been tested on an NVidia GTX Titan and compiled using gcc 4.7 and CUDA 5.5. There is only one execution mode for GPUs which does require memory copies between the host and device. However, these memory copies have not been included in the results to ensure a fair comparison between the Xeon Phi and the NVidia GPU. The CUDA kernel for computing the simple test case on the GPU is shown in Listing 2.

Listing 2: CUDA kernel implementation of the simple test case.

```
__global__ void simple(RNG *r, double *total, int N) {
  int i = ((blockIdx.x * blockDim.x)+threadIdx.x);
  RNG rng = r[i];
  double subtotal = 0.0;
  for(int n = 0; n < N; n++) {
    subtotal += uniform(rng);
  }
  atomicAdd(total, subtotal);
}
```

CPU Model Platform	CPUs	Cores	L2 Cache (MBytes)	CPU Clock (GHz)	Operating System
SMD Blade Intel Xeon 1x8 Core	1	8	20	3.10	Ubuntu
SMD Blade AMD Bulldozer 2x16 Core	2	32	2	2.20	Ubuntu
SMD Blade Intel Xeon Phi 5110P	1	60	0.5	1.05	CentOS
SMD Blade GTX Titan	1	2688	1.5	0.88	Ubuntu

Table 1: CPU, GPU and co-processor properties for the systems used for benchmarking.

5 Selected Performance Results

The benchmarks have been executed on each of the different test machines for sizes of $N = 10^8..1.5\text{x}10^9$. Because of memory limitations, not all of the computing platforms tested were able to complete the benchmarks for all system sizes. The memory requirements have the most noticeable impact on the Xeon Phi and the Kepler GPU. The Xeon Phi has 8GB of memory but some of this is required to run the operating system on the device and the NVidia GTX Titan has only 6GB of memory. The CPU machines had no such problems with available memory ranging from 16GB to 128GB.

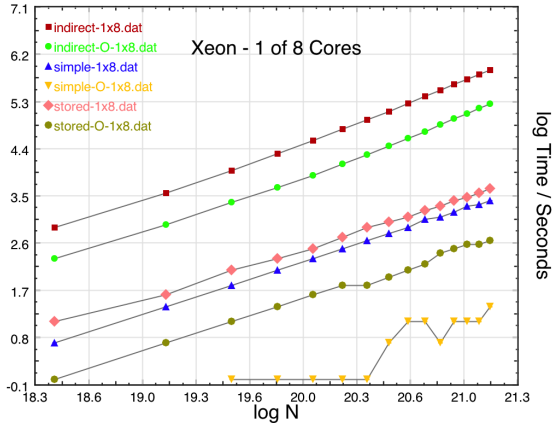


Figure 5: Xeon 8-core 3.10 GHz Blade

Figure 5 shows the benchmark results executed on a single core of a 3.10 GHz Intel Xeon. The results are largely as expected, the indirect addressing (with and without optimisations) have the slowest performance because of the random access restricting the CPU's ability to cache memory access. It also shows the limitations of the timing which cannot accurate measure the time taken for the optimised simple benchmark which is not restricted by memory access.

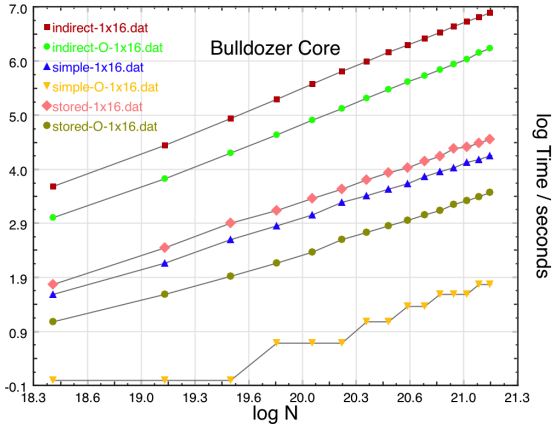


Figure 6: Bulldozer 2.20 GHz Blade

Figure 6 shows the performance of the same benchmarks executed on a single core of an 2.20 GHz AMD Bulldozer. This plot shows similar trends to the Intel Xeon from Figure 5. Comparing these two plots it can be seen that the performance of the Bulldozer is measurably lower than the Intel Xeon, not surprising given the difference in clock speed.

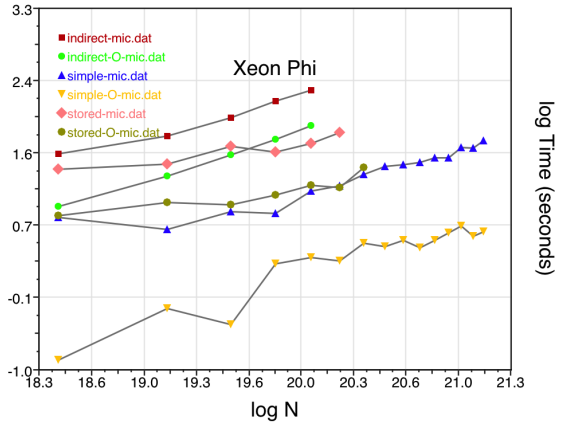


Figure 7: Xeon Phi in native execution mode.

Figure 7 shows the TBB implementations of these benchmarks executed on the Intel Xeon Phi. The full performance plots for the stored and indirect benchmarks are not available due to the memory limitations of the Xeon Phi. This implementation is executed on all available processors of the Xeon Phi. It should be noted that this implementation does not explicitly make use of the vector processing units available in the Xeon Phi cores.

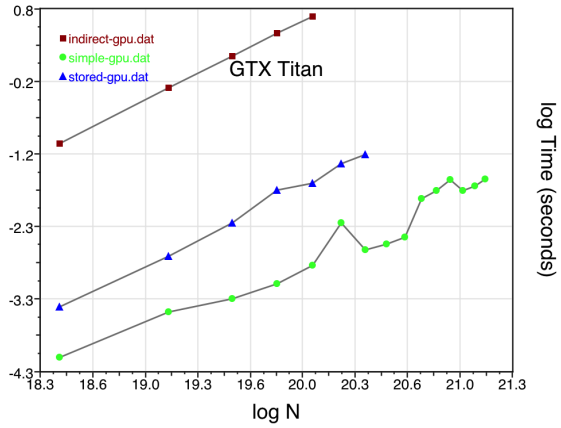


Figure 8: Xeon Blade equipped with GTX Titan.

Figure 8 shows the benchmarks executed on an Nvidia GTX Titan. The GPU can only test a limited set of benchmarks due to the memory constraints of the device. As with previous plots the performance of the simple benchmark is hard to measure due to the very short execution time. The largest benchmark at $1.5\text{x}10^9$ can be computed on the GTX Titan in less than 0.2 seconds.

While the Intel Xeon and AMD Bulldozer show similar trends in performance, the single Xeon core completes the benchmark tests between 1.5x faster (simple case) to 2.5x faster (indirect and stored case). Compared to the single AMD Bulldozer core the Xeon Phi (using all 60 physical cores) can compute the benchmarks between 3x faster (simple case) to 20x faster (indirect case). Overall the GTX Titan provides the best performance for the benchmarks completing the simple case 35x faster than a single Bulldozer core and 60x faster for the stored and indirect test cases. Although the GPU and Xeon Phi provide performance improvements it should be remembered that they are limited to the size of system they can compute for the stored and indirect test cases due to the limited memory available on the devices.

Table 2: Comparison of price and benchmark results (generating $5x10^9$ random numbers). Prices are approximated from online information 2013.

Platform	Price (NZD)	Simple	Stored	Indirect
Bulldozer	≈ $950	2s	11s	132s
Xeon	≈ $2000	1s	5s	49s
Xeon Phi	≈ $3400	1.4s	3.3s	6.7s
GTX Titan	≈ $1800	0.05s	0.2s	2.2s

6 Developmental Directions

We have managed to find a benchmark algorithm and set of sizes that can practically be run on all the platforms we have discussed. There are of course many possible ways to approach the parallelisation of even these simple loops however. We have focused on those that ought to have favoured the individual platforms - using Intel proprietary compiler and tools on the Xeon Phi and NVidia's CUDA and associated tools on the Kepler GPU. The GNU compiler set and its virtual register approach to compiler level optimisation still delivers good performance on a range of conventional CPU architectures. Various threading libraries including pThreads and Intel's Thread Building Blocks can also be used.

Our overall impression of these devices and tools is that the automated parallelism problem is still far from being solved. NVidia's Kepler GPUs and its likely successors do well because they limit the scope of the parallelisation problem that they commit to tackle. GPUs are extremely good at fine grained data parallel applications and algorithms and although the number of multi processors and floating point units is improving the memory communications structure will still favour problems of a data-parallel nature. GPUs do well precisely because programmers know what they are good at, they are not trying to support all parallel models and tools exist to optimise against their strengths.

Multiple cores in conventional CPUs offer considerable power for multi threading application programs. The performance attainable with Intel's hyper-threading architecture on 6-core or 8-core Xeons is impressive and exploitable at the application level. AMD's Bulldozer processor with its 16-cores is also impressive and although it does not match the Xeon performance it perhaps comes close on price performance.

The Intel Xeon Phi many-integrated core (MIC) is a valiant step in the right direction, but we and others do seem to be failing to make it deliver against its potential. Perhaps its simply offers just to many cores (circa 60) against the communications and memory support infrastructure that comes with it at present? It certainly offers some interesting speed ups but nothing close to what one might hope from 60 substantive Knights Corner cores. We found that it was very difficult to tie together all the necessary software technologies to program it. NVidia's CUDA ecosystem is not trivial to program either but does deliver significantly more worthwhile performance. It is not an easy matter to quantify programmer effort against attainable performance but in the present world where generally programmers are a lot more expensive than raw hardware this is an important consideration. It is our opinion that Intel or its support community will have to put a lot more effort into tools, example codes, and other software ecosystem support before the Xeon Phi accelerator and it s hoped for successors are more widely adopted.

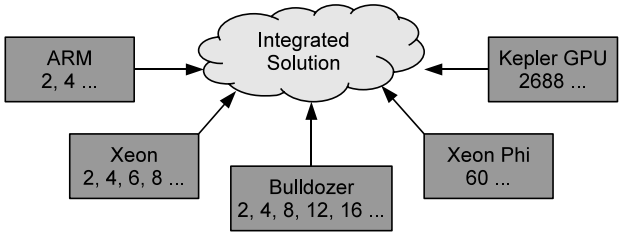


Figure 9: Development directions - multi-core CPU and parallel accelerator designs converging on an integrated design.

We do hope this will happen as it seems that the general development of accelerators will be an important part of parallel computing and the battle against heat density and chip packaging limitations for some time to come.

Figure 9 shows how different vendors are approaching the problem of introducing exploitable cores from different directions. NVidia's approach has been based on introducing a large number of lightweight cores capable of supporting fine-grained data-parallelism. Further acceleration may come by incorporating additional controllers and based on ARM CPU architectural units. Intel's traditional approach has been to increase the number of conventional CPU cores available within the CPU itself and is now accelerating these even further by adding even more accelerator cores in the form of devices like the MIC.

An important notion in parallel computing is and ha s been for some time that of "balance" in terms of the ratios between raw computational performance (Gigaflops) and memory size and indeed access rates to keep processors fed. Looking back over the last twenty years, platforms that have a ratio of around 1 MFlops performance to 1MByte storage were the norm at the end of the 1980s. This ratio has continued approximately with present desktops offering approximately similar ratios. This indicates that our benchmark targets roughly the right range and that this is a good indicative range of the performance regime that CPUs and accelerated CPUs must target.

While applications vary and can target all aspects of the hierarchy discussed in Section 1, it seems likely that vendors need to put more effort into tackling the memory and communications support infrastructure if accelerator architectures based on conventional CPU cores are to successfully counter other approaches.

7 Summary & Conclusions

We have described the present state of the art in terms of conventional multi-core CPUs and accelerator devices and have listed typical numbers of cores, clock rates and memory support and cache sizes. We have experimented with specific instances of all these devices and have attempted to assess their relative strengths and weaknesses by a detailed benchmarking exercise.

We formulated and have given code details of a series of simple yet informative benchmark codes in C that exercise both raw computational performance as well as memory support structure and which use a portable random number generator to allow a practicable and fair comparison of all devices. We believe this is the first time that a quantitative comparison has been reported on **all** of these platforms together.

We have found that while CPUs using conventional cores are easily programmed an indeed yield an impressive absolute and speedup performance, the fine grained GPU accelerator yields the best speedup. It also yields the best price performance too. The Intel Xeon Phi accelerator is an exciting development and it typifies Intel's alternative approach to NVidia. However despite our expending around an order of magnitude more programming effort to try to obtain good performance from the Xeon Phi, it has not yet delivered its hoped for performance.

We believe it is important to tackle the many cored problem from this direction, as Intel is supporting, but that likely more effort needs to be expended on communications and memory support infrastructure to ensure the many cores can be practicably exploited at the application level.

There is scope for combining multiple accelerator devices together to service a single application. Use of a GPU and Xeon Phi together accelerating a single host CPU is technically feasible and may yield some interesting insights on balance, device specialisation and the future potential for hybrid systems.

Our benchmarks reported here are simplistic - deliberately so to ensure all platforms and their associated optimisation tools had a fair chance. There is therefore scope for a richer set of benchmark tests - particularly those exercising more complex communications patterns and commonly used data structures and algorithms including linear algebra operations but also regular meshes and irregular graph network problems.

Finally, we believe the accelerator approach will continue to be an important one, and that both the intel approach of integrating together conventional CPU cores and the NVidia hierarchical approach to fine grained cores are both necessary for driving development of the parallel software ecosystem. When these two approaches meet together in terms of core numbers is likely to be an interesting time for parallelism.

References

Bailey, D., Barscz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V. & Weeratunga, S. (1994), The nas parallel benchmarks, Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, CA, USA.

Bischof, C. H. & Dongarra, J. J. (1989), A linear algebra library for high-performance computers, *in* G. F. Carey, ed., 'Parallel Supercomputing: Methods, Algorithms and Applications', Wiley, chapter 4, pp. 45–55.

Blumofe, R., Joerg, C., Kuszmaul, B., Leiserson, C., Randall, K. & Zhou, Y. (1995), Cilk: An efficient multithreaded runtime system, *in* 'Symp. Principles and Practice of Parallel Programming', ACM, pp. 207–216.

Butler, M., Barnes, L., Sarma, D. D. & Gelinas, B. (2011), 'Bulldozer: An approach to multithreaded compute performance', *IEEE Micro* **31**(2), 6–15.

Cantrill, B. (2006), 'Hidden in plain sight', *System Performance* **4**, 26–36.

Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D. & McDonald, J. (2001), *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers Inc.

Demmel, J., Dongarra, J., DuCroz, J., GreenBaum, A., Hammarling, S. & Sorensen, D. (1988), A project for developing a linear algebra library for high-performance computers. ANL-MCS-P37-1288 Preprint.

Giles, M. (2009), Notes on CUDA implementation of random number genertors. Oxford University.

Grove, D. A. & Coddington, P. D. (2005), 'Communication benchmarking and performance modelling of mpi programs on cluster computers', *J. Supercomput.* **34**(2), 201–217.

Hawick, K. A. & Playne, D. P. (2013), Parallel algorithms for hybrid multi-core cpu-gpu implementations of component labelling in critical phase models, *in* 'Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'13)', number CSTN-177, WorldComp, Las Vegas, USA, p. PDP3297.

IEEE (1995), *IEEE Std. 1003.1c-1995 thread extensions*.

Intel (2010), *Intel(R) Threading Building Blocks Reference Manual*, Intel.

Intel (2013), Intel 64 and ia-32 architectures optimization reference manual, Technical report, Intel.
URL: *http://www.intel.com/*

Kleidermacher, D. N. (2008), 'Multicore software development: Fact and fiction', *Embedded Systems Design* **November**, 1–5.

Knauerhase, R., Cledat, R. & Teller, J. (2012), For extreme parallelism, your os is sooooo last-millennium, *in* 'Proc. 4th Usenix Conf. on Hot Topics in Parallelism (HotPar'12)'.

Leist, A., Playne, D. P. & Hawick, K. A. (2009), 'Exploiting Graphical Processing Units for Data-Parallel Scientific Applications', *Concurrency and Computation: Practice and Experience* **21**(18), 2400–2437. CSTN-065.

Matsumoto, M. & Nishimura, T. (1998), 'Mersenne twistor: A 623-diminsionally equidistributed uniform pseudorandom number generator', *ACM Transactions on Modeling and Computer Simulation* **8 No 1.**, 3–30.

McMahon, F. H. (1986), Livermore fortran kernels: A computer test of numerical performance range, Technical Report UCRL-53745, Lawrence Livermore National laboratory, Livermore, CA, USA. NTIS Report DE87009360.

NVIDIA (2012), *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110*.
URL: *http://www.nvidia.com/*

Oskin, M. (2008), 'The revolution inside the box', *Communications of the ACM* **51**(7), 70–78.

Patterson, D. A. & Hennessy, J. L. (2009), *Computer Organization and Design - The Hardware/Software Interface*, number ISBN 978-0-12-374493-7, Morgan Kaufmann.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. (2007), *Numerical Recipes - The Art of Scientific Computing*, third edn, Cambridge. ISBN 978-0-521-88407-5.

Reinders, J. (2007), *Intel Threading Building Blocks: outfitting C++ for multi-core processor parallelism*, number ISBN 978-0596514808, 1st edn, O'Reilly.

Saule, E., Kaya, K. & Catalyurek, U. V. (2013), Performance evaluation of sparse matrix multiplication kernels on intel xeon phi, arXiv 1302.1078v1, Ohio State University.

Sutter, H. & Larus, J. (2005), 'Software and the concurrency revolution', *Queue* **3**(7), 54–62.

TOP500.org (n.d.), 'TOP 500 Supercomputer Sites', `http://www.top500.org/`. Last accessed November 2010.

Simulating and Benchmarking the Shallow-Water Fluid Dynamical Equations on Multiple Graphical Processing Units

D.P. Playne **K.A. Hawick** **M.G.B. Johnson**

Computer Science, Institute of Natural and Mathematical Sciences
Massey University – Albany
North Shore 102-904, Auckland, New Zealand
Email: {d.p.playne, k.a.hawick, m.johnson}@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

Abstract

The shallow-water model equations provide a simple yet realistic benchmark problem in computational fluid dynamics (CFD) that can be implemented on a variety of computational platforms. Graphical Processing Units can be used to accelerate such problems either singly using a data parallel decompositional scheme or with multiple devices using a domain decompositional approach. We implement the SW equations on a range of modern GPUs with both parallel schemes and report on the typical performance. We compare integer optimised GPUs and very modern floating-point intensive GPU devices such as NVidia's Kepler K20X, and also investigate different m-GPU communication methods for geometric decompositions. We give detailed performance results and a summary of the main parallelisation issues.

Keywords: shallow water system; weather simulation; climate simulation; parallel computing; single GPU; multiple GPU.

1 Introduction

Computational Fluid dynamical algorithms(Tritton 1988, Griebel, Dornseifer & Neunhoeffer 1998) remain the central core needed for both numerical weather prediction(Barry & Chorley 1989) and climate simulations(Hurrell, Meehl, Bader, Delworth, Kirtman & Wielicki 2009). These problems have been central to high performance computing since its inception with the ENIAC and other supercomputers in the 1950s(Dyson 2012). Modern weather forecast codes and climate analysis codes have become very sophisticated in modern times, often running ensembles of systems to attain appropriate statistical accuracy. Nevertheless the core of such codes is still a numerical integration of the atmosphere as a fluid and sometimes also a coupled ocean model system. In both cases the system is treated as a fluid with coupled cells of material each approximating a spatial region of air or ocean(C.A.J.Fletcher 1991*a*, C.A.J.Fletcher 1991*b*). A great many different numerical schemes have been employed in such codes and it is quite difficult to implement a complete such package as a benchmark(Bailey, Barscz, Barton, Browning, Carter, Dagum, Fatoohi, Fineberg, Frederickson, Lasinski, Schreiber, Simon, Venkatakrishnan & Weeratunga 1994).

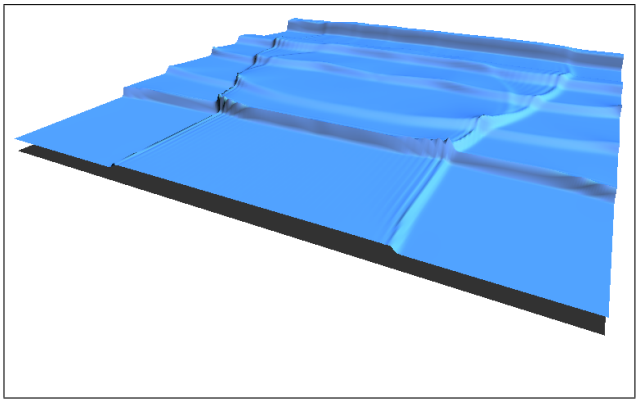


Figure 1: Visualisation of waves in a Shallow Water model simulation.

The shallow-water (SW) system of equations(Randall 2006) model a greatly simplified such problem and provide a more practicable benchmark algorithm and code that can be readily ported across different computational platforms, and most usefully for our work in this present paper, the SW equations also lend themselves to a range of different parallel decompositional schemes. As the name suggests, the SW system models a shallow fluid system that is somewhere between two and three dimensional. A few simplified fluid layers are used to approximate the bulk behaviour and simplify the calculations. Although in such simulations much of the details needed to study for example turbulence(W.D.McComb 1990) are lost, the SW model tends to work surprisingly well for thin-layer fluids(Hawick 2011) such as the planetary atmosphere or for certain shallow oceanic problems such as particular bays(Bailey 2010, Martinez, Campbell, Annable & Kiker 2008). Realistic short to medium range weather models make intensive use of initial condition values for the numerical integration(Hawick 1991) based on observational data form many sources such as satellites(Xiao, Zou, Pndeca, Shapiro & Velden 2002). Climate codes are more governed by statistical energy(Peixato & H.Oort 1984) and system-wide coupling effects and exhibit less memory concerning specific initial condition details. Benchmarking and optimising the performance of the data assimilation scheme(Hawick, Bell, Dickinson, Surry & Wylie 1992) is a separable problem however and we do not tackle it in this present paper. Instead we focus on the computational integration(Playne & Hawick 2011) as typified by the needs of a model such as the SW system.

Although sophisticated spectral techniques(Barros

& Kauranne 1991, Tett 1991) are increasingly being applied to weather and climate problems, the classic parallel strategy for computational fluid dynamics(Hawick, Bogucz, Degani, Fox & Robinson 1995) is that of a geometric data parallelism(Hawick & Playne 2011), whereby the mesh points of the equations are spread geometrically across a set of processors or processing cores. This strategy is ideal for a single Graphical Processing Unit (GPU) and modern GPUS(Leist, Playne & Hawick 2009) have an impressively large number of such cores. However not all such devices have the same degree of support for floating point calculations(Johnson, Playne & Hawick 2013), and some devices that have many cores have rather poorer FPU support of those cores. There is therefore an important tradeoff space of FPU support and number of cores and of course cost to be considered.

Additionally, schemes like the SW model which involves multiple stages of advection also offer a task decompositional option(Vu, Cats & Wolters 2008) whereby multiple GPUs are used to accelerate the same CPU host processor in tackling different parts of the numerical update. This is a particularly interesting strategy that can make use of the modern GPU capability of communicating data with one another directly *without* passing communications through the hosting CPU.

Modern multi-core processors and GPUs lend themselves well to computationally intensive calculations of this sort(Schmidt, Berzins, Thornock, Saad & Sutherland 2013). Many of the current Top 500 supercomputers in the world use GPUs to accelerate individual nodes. The regime of multiple GPU accelerators per node is however still not well explored. Computational Fluid Dynamics remains a highly significant problem for real supercomputer installations(Zaspel & Griebel 2012). CFD problems have also been formulated as lattice-gas and automaton model systems(Johnson, Playne & Hawick 2010, Lyes, Johnson & Hawick 2012). This approach can make use of integer-optimised devices but in this present paper we focus of traditional CFD solutions using the solution of partial differential equations by numerical integration(Micikevicius 2009) and floating-point intensive processing elements.

The SW system itself has been studied before for many parallel platforms including a single GPU accelerator per CPU(Vinas, Lobeiras, Fraguela, Arenaz, Amor, Garcia, Castro & Doallo 2013). We however are able to consider the tradeoff space for very recent GPU models and also for multiple GPUs including the GPU-GPU direct data communication capability. We make use of NVidia's Compute Unified Device Architecture (CUDA) software(NVI 2012) in this present work. CUDA software can be run on a large range of different GPU models and we report on a selection of these along with discussion of the implications for CFD simulation problems.

Our article is structured as follows: In Section 2 we summarise the Shallow Water model formulation that we investigate. We describe our parallelisation strategy for single and multiple GPUs in Section 4 and give some selected performance results for various devices in Section 5. We give a discussion of the results and offer some tentative conclusions in Section 6.

2 Shallow Water Model Formulation

The Shallow Water equations (SW) are a set of partial differential equations that can be derived from the more general Navier-Stokes equations. In the SW the horizontal length scale is considered to be much greater than the vertical length scale and a hydrostatic pressure along the direction of gravity. The vertical velocity can be removed from the Navier-Stokes equations by integrating vertically which allows a three-dimensional fluid problem to be turned into a two-dimensional height-field problem (Muller, Stam, James & Thurey 2008).

The SW describe two conditions of the conservation of mass and the conservation of momentum. This is expressed as an advection-diffusion problem.

A shallow-water system can be represented by a fluid height- and velocity-field.

$$\frac{\partial h}{\partial t} = -h\nabla \cdot \mathbf{v} - (\nabla h)\mathbf{v} \quad (1)$$

$$\frac{\partial \mathbf{v}}{\partial t} = g\nabla h - (\nabla \mathbf{v})\mathbf{v} \quad (2)$$

The fluid in this model is stored as a staggered grid where the height variable h represents the height of the fluid at the centre of each discrete cell. The horizontal velocity ($\mathbf{v}$) is represented by two variables vx and vy which are the velocities on the edges of the cell in the x- and y-dimensions respectively. This staggered grid is shown in Figure 2 where the dots represent the location of the height variable and the lines represent the horizontal velocities in the x- and y-dimensions. Staggered grids are commonly used for fluid simulations as they avoid some of the instabilities caused by a discrete co-located grid.

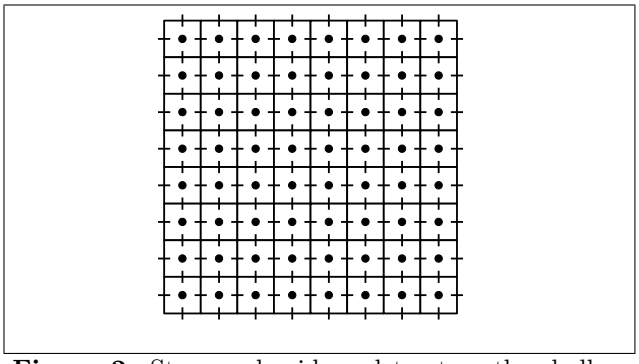


Figure 2: Staggered grid used to store the shallow water system. The dots represent the location of the height field points and while the vertical and horizontal bars represent the velocity field points.

The method of simulating the SW used in this project is based on the description in (Muller et al. 2008). This method uses an explicit time-integration scheme to avoid the problem of solving a system of linear equations. Updating the fluid involves two separate stages - first the fields are advected using the velocity field and then the accelerations of the fields are computed. This process is shown in Algorithm 1.

A semi-Lagrangian method is used to compute the advection steps within the field. Essentially this involves performing a backwards trace of an imaginary particle at each lattice point. This must be performed separately for each of the lattice points representing the height and velocity fields. For each lattice point the velocity field at that point is calculated. This may

Algorithm 1 Update algorithm for the SW model.

```
for All steps do
  advect(h, v)
  advect(vx, v)
  advect(vy, v)
  update(h, v)
  update(vx, h)
  update(vy, h)
end for
```

require interpolation of velocity variables due to the staggered grid. This is illustrated in Figure 3.

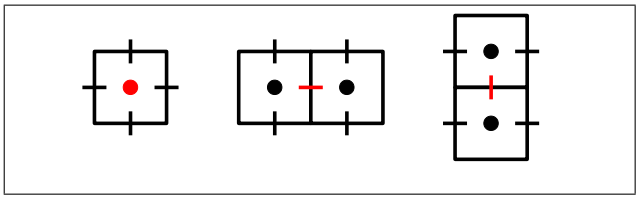


Figure 3: Interpolation of the velocity field variables for the three different lattice points - height (left), vx (centre) and vy (right).

Once the appropriate velocity values have been interpolated to give the velocity field at the desired point in the grid, the velocity is used to trace an imaginary particle backwards in time to advect the field. Once the particle has been traced backwards in time (see Equation 3), the value of the field must be calculated at the exact point. This is performed using bi-linear interpolation which is bounded by the existing values of the field and helps to ensure stability.

$$\mathbf{x}_{t-\triangle t} = \mathbf{x} - \mathbf{v}\triangle t \tag{3}$$

After the advection steps have been completed the fields are updated. The update of the height field depends the divergence of the velocity fields and the acceleration of the velocity fields is given by the gradient of the fluid height (above 0). This total fluid height depends on both the fluid height and the height of the ground to correctly handle non-constant ground planes.

The border conditions of the simulation are simple reflecting boundaries. These boundaries can be implemented by enforcing the condition that the velocity across the border is zero (fluid cannot flow through the wall) and mirroring the height of the fluid across the boundary. The velocity along the direction of the wall is unaffected by the boundary conditions and the fluid can flow along the wall with no friction.

This is a brief outline of the updating process of a simple shallow-water simulation based on the description in (Muller et al. 2008). This article considers how such a simulation can be implemented on an m-GPU (multi-GPU) system.

3 m-GPU Technology

Released late last year the Nvidia Tesla K20X is the most powerful single-GPU released to date. This Kepler architecture GPU contains 2688 CUDA cores, 6GB of GDDR ram and supports Dynamic Parallelism and Hyper-Q. With a peak double precision floating point performance of 1.31 TFLOP/s it is over twice as fast as the previous generation Tesla M2090 (666.1 TFLOP/s) despite only having an additional 10 watts TDP.

The GPU in this device is the Kepler GK110 which contains a number of NVIDIA's new generation Streaming Multiprocessor (SMX) units shown in Figure 4. These SMX units contain a number of improvements over those in the GK104, most significant for scientific applications is the additional double precision processing units. For reference the architecture of the previous generation Streaming Multiprocessor (SM) units in a Fermi GPU are shown in Figure 5.

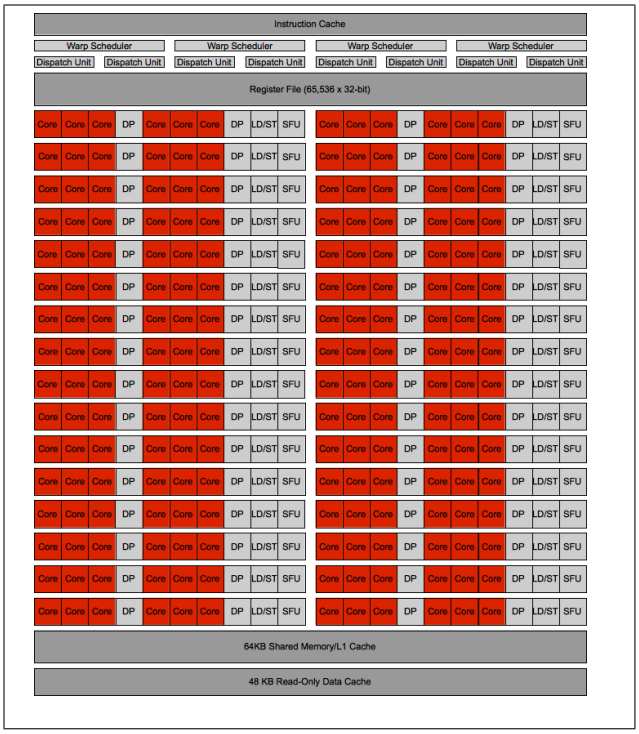


Figure 4: Architecture of an SMX unit from a GK110 GPU which powers cards such as the K20X and the GeForce GTX Titan.

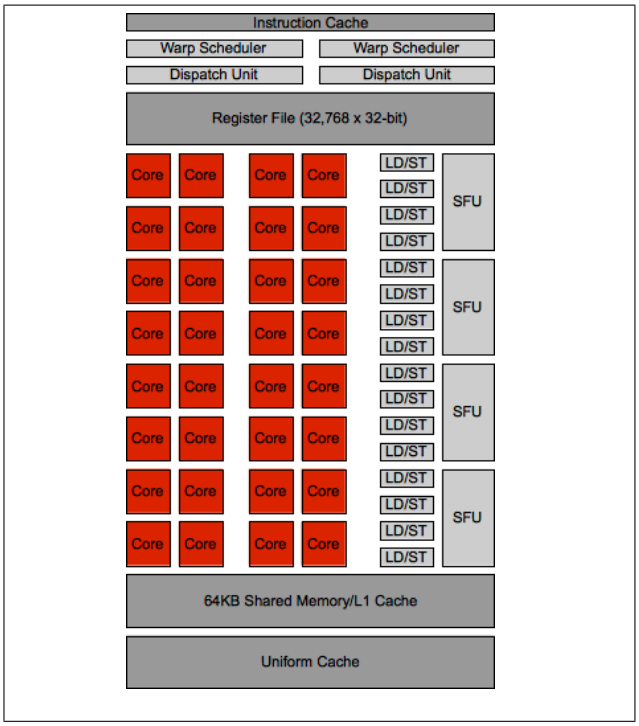


Figure 5: SM unit architecture from a Fermi architecture GPU in cards such as the GTX590.

Transfer Methods

There are two competitive options for data transfer between devices in an m-GPU program. These are Direct Transfer and Direct Access which are peer-to-peer data transfer methods available with GPUDirect 2.0. These two methods require two GPUs of Fermi-architecture or later with peer-to-peer memory access enabled and both rely on Unified Virtual Addressing (UVA). UVA maintains a single address space for all memory allocated on the host and on any GPU devices. This allows any memory access or copy to identify the memory location of a memory address whether it is on the host or on a GPU.

Direct Transfer allows the host to initiate a memory copy from a source the device memory of one GPU to a destination in another device. A Direct Transfer memory copy is faster than the old method of memory transfer between devices which had to copy the data into an area of host memory and then copy it into the second device. Direct Transfer eliminates this unnecessary additional memory transfer and the need for a host memory buffer. This method of device-device communication is shown in Figure 6.

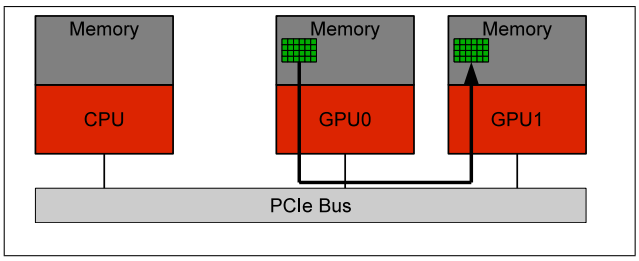


Figure 6: Direct Transfer method of communication where data is copied directly from the memory of one device to another.

Direct Access eliminates the need for host initiated memory copies altogether by allowing threads from one GPU to access memory on another GPU. This Non-Uniform Memory Access (NUMA) is expected to be less efficient for transferring large contiguous blocks of data between two GPU devices but can have advantages for the transfer of border information as the memory copies can be removed entirely. This method of data access is shown in Figure 7.

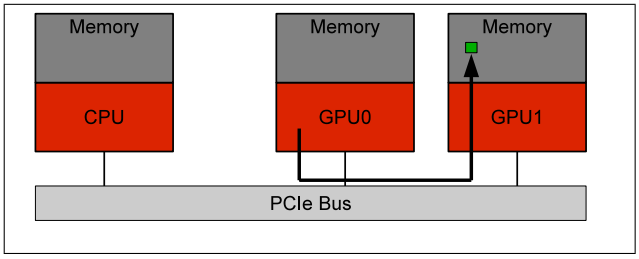


Figure 7: Direct Access method of communication where data stored in one device is read directly by a thread from another device.

Unified Virtual Addressing (UVA) defines a single address space in which all memory addresses are defined for both the CPU and GPU memory. This allows the physical location of a memory address to be identified by the system. This feature is required for Direct Access as if there was not a single address space then the threads would not be able to identify which device's memory a value should be read from. Direct Transfer does not require UVA but it does simplify the `cudaMemcpy` calls for a Direct Transfer copy as the programmer can simply specify two addresses and the system can work out where those memory addresses are stored. Previously the programmer had to use `cudaMemcpyPeer`, specify the source and destination addresses as well as the GPU devices they were stored on.

4 GPU Implementation Method

Domain decomposition is used to split the simulation of the Shallow Water model across multiple GPU devices. In the case describe here where the system is split between two GPU devices, each device is allocated half of the system. The devices are responsible for storing and updating their half of the system. The system is split in the y-dimension because then the bordering cells in the lattice are stored in contiguous memory. It should be noted that splitting the simulation across a different number of devices may have other optimal decomposition methods, the optimal method to used will depend on the number of devices and is not discussed further here.

During the updating of the system, the data on the borders of each device's domain must be exchanged in order for the simulation to be correct. This exchange of bordering information must be performed several times during the update (see Algorithm 1). The advection of the fields is performed using the current velocity field (Muller et al. 2008) and the new values computed for the borders of each domain must exchanged. This exchange must also be performed after the update of the height field (as it is required for the update of the velocities) and finally after the velocities have been updated. The bordering areas of the two lattice domains are shown in Figure 8.

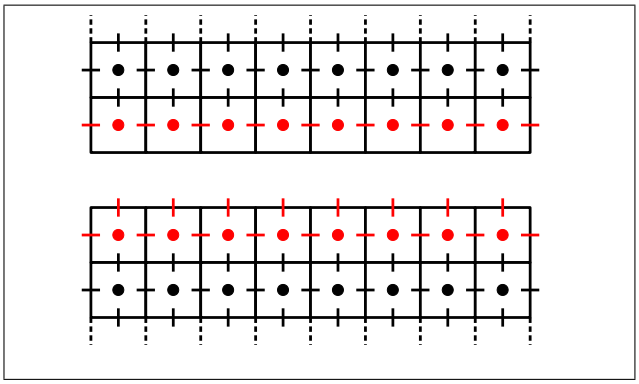


Figure 8: Bordering information that must be exchanged between the GPU devices.

Direct Transfer Implementation

The first implementation of the m-GPU shallow water model uses the Direct Transfer method to exchange bordering information. Performing a direct memory copy from one device to another is simple in the more recent version of CUDA. All that is required is a simple `cudaMemcpy` call using the `cudaMemcpyDefault` flag. Because the memory addresses are defined with Unified Virtual Addressing (UVA), CUDA automatically works out where the source and destination memory areas are stored and copies the data between them. If these two areas are in the device memory of two different GPUs it will automatically use the Direct Transfer method.

Listing 1 shows a code snippet of the advection stage of the SW simulation. The three fields `h`, `vx` and `vy` are advected using the current velocity field by each GPU device. The devices are synchronised to ensure that the computation is completed before the bordering information is exchanged which is then performed by a series of call to `cudaMemcpy`.

Listing 1: Direct Transfer Implementation of the advection stage of the simulation. The three fields of the simulation are numbered to show which device they belong to - e.g. `h1` is the height field on device 1, `h2` is the height field on device 2.

```
//Advection
cudaSetDevice(0);
advectH <<<  ... >>>(vx1, vy1, h1, ...);
advectVX <<< ... >>>(vx1, vy1,     ...);
advectVY <<< ... >>>(vx1, vy1,     ...);

cudaSetDevice(1);
advectH <<<  ... >>>(vx2, vy2, h2, ...);
advectVX <<< ... >>>(vx2, vy2,     ...);
advectVY <<< ... >>>(vx2, vy2,     ...);

//Synchronise threads
cudaSetDevice(0);
cudaThreadSynchronize();
cudaSetDevice(1);
cudaThreadSynchronize();

//Exchange H
cudaMemcpy(&h1[Y2*X], &h2[Y2*X],
           X*sizeof(real), cudaMemcpyDefault);
cudaMemcpy(&h2[(Y2-1)*X], &h1[(Y2-1)*X],
           X*sizeof(real), cudaMemcpyDefault);

//Exchange VX
cudaMemcpy(&vx1[(Y2)*X1], &vx2[(Y2)*X1],
           X1*sizeof(real), cudaMemcpyDefault);
cudaMemcpy(&vx2[(Y2-1)*X1], &vx1[(Y2-1)*X1],
           X1*sizeof(real), cudaMemcpyDefault);

//Exchange VY
cudaMemcpy(&vy1[(Y2)*X], &vy2[(Y2)*X],
           X*sizeof(real), cudaMemcpyDefault);
cudaMemcpy(&vy2[(Y2-1)*X], &vy1[(Y2-1)*X],
           X*sizeof(real), cudaMemcpyDefault);
```

The disadvantage of the Direct Transfer method is that the GPU devices will be idle during the memory transfer. Unlike the copy through host, CUDA does not currently support concurrent computation and communication for peer-to-peer memory transfer.

Direct Access Implementation

The Direct Access method has no need for any calls to `cudaMemcpy` as any kernel that requires data stored in the other GPU device's memory can simply read it. This does require each thread to have pointers to the fields on both GPUs. Using UVA, CUDA can automatically work out where the memory is stored and each thread will either read the value straight from device memory or read it from the other device using Direct Access across the PCIe bus. This has been implemented in the simulation by using a simple `if` statement to read the field value from the array stored on the same or other GPU.

This is implemented in all of the kernels but the height advection kernel is shown as an example in Listing 2. This kernel is the same for both GPU devices and simple updates on half of the field as determined by `y` and `y_max`. If any address has a y index of less than `Y2` the thread will read the value from the array on the first device, if it is greater than or equal to `Y2` it will be read from the second device. This address comparison only needs to be performed when neighbouring values are read as a thread will never be launched on one GPU to process a cell stored on the other device.

Listing 2: Direct Access Kernel Implementation.

```
__global__ void advectH(real *h1_1, real *h1_2,
                        real *vx_1, real *vx_2,
                        real *vy_1, real *vy_2,
                        real *h2, int y,
                        int y_max) {
  int iy = (blockIdx.y * blockDim.y) +
           threadIdx.y + y;
  int ix = (blockIdx.x * blockDim.x) +
           threadIdx.x;

  if(iy < y_max) {
    int ym1 = max(iy - 1, 0);
    int xm1 = max(ix - 1, 0);
    int xp1 = min(ix + 1, X1-1);
    int yp1 = min(iy + 1, Y1-1);

    real vx_yx   = (iy < Y2) ?
         vx_1[iy*X1+ ix]:vx_2[iy*X1+ ix];
    real vx_yxp1 = (iy < Y2)  ?
         vx_1[iy*X1+xp1]:vx_2[iy*X1+xp1];

    real vy_yx   = (iy < Y2) ?
         vy_1[ iy*X+ix]:vy_2[ iy*X+ix];
    real vy_yp1x = (yp1 < Y2) ?
         vy_1[yp1*X+ix]:vy_2[yp1*X+ix];

    real ax = ix-((vx_yx+vx_yxp1)/2.0)*dt;
    real ay = iy-((vy_yx+vy_yp1x)/2.0)*dt;

    int nx = floor(ax);
    int ny = floor(ay);

    ax = ax - nx;
    ay = ay - ny;

    xp1 = min(nx+1, X-1);
    yp1 = min(ny+1, Y-1);

    real h_yx     = (ny < Y2)  ?
         h1_1[ny*X +nx] :h1_2[ny*X +nx];
    real h_yxp1   = (ny < Y2)  ?
         h1_1[ny*X +xp1]:h1_2[ny*X +xp1];
    real h_yp1x   = (yp1 < Y2) ?
         h1_1[yp1*X+nx] :h1_2[yp1*X+nx];
    real h_yp1xp1 = (yp1 < Y2) ?
         h1_1[yp1*X+xp1]:h1_2[yp1*X+xp1];

    h2[iy*X + ix] = h_yx*(1.0-ax)*(1.0-ay)+
                    h_yxp1*(ax)*(1.0-ay)+
                    h_yp1x*(1.0-ax)*(ay)+
                    h_yp1xp1*(ax)*(ay);
  }
}
```

The performance of both these implementations are compared to each other and to a single-GPU implementation of the same simulation.

5 Performance Results

These two m-GPU implementations have been tested on several different m-GPU systems with different generations of GPU device. The GPUs tested are the Fermi architecture GeForce GTX590 and the Kepler architecture GTX690 and two GTX780s. In addition to these GeForce cards the simulations are tested on two Kepler architecture Tesla compute card K20Xs. These systems are running CUDA version 5.5 which supports NVIDIA GPUDirect 2.0, the technology that provides the Direct Transfer and Direct Access methods.

The implementations have been tested for a range of system sizes from 64^2 to 8192^2 (GPU device memory allowing). These experiments compare both the computational performance of the different GPU devices as well as their support for device-device com-

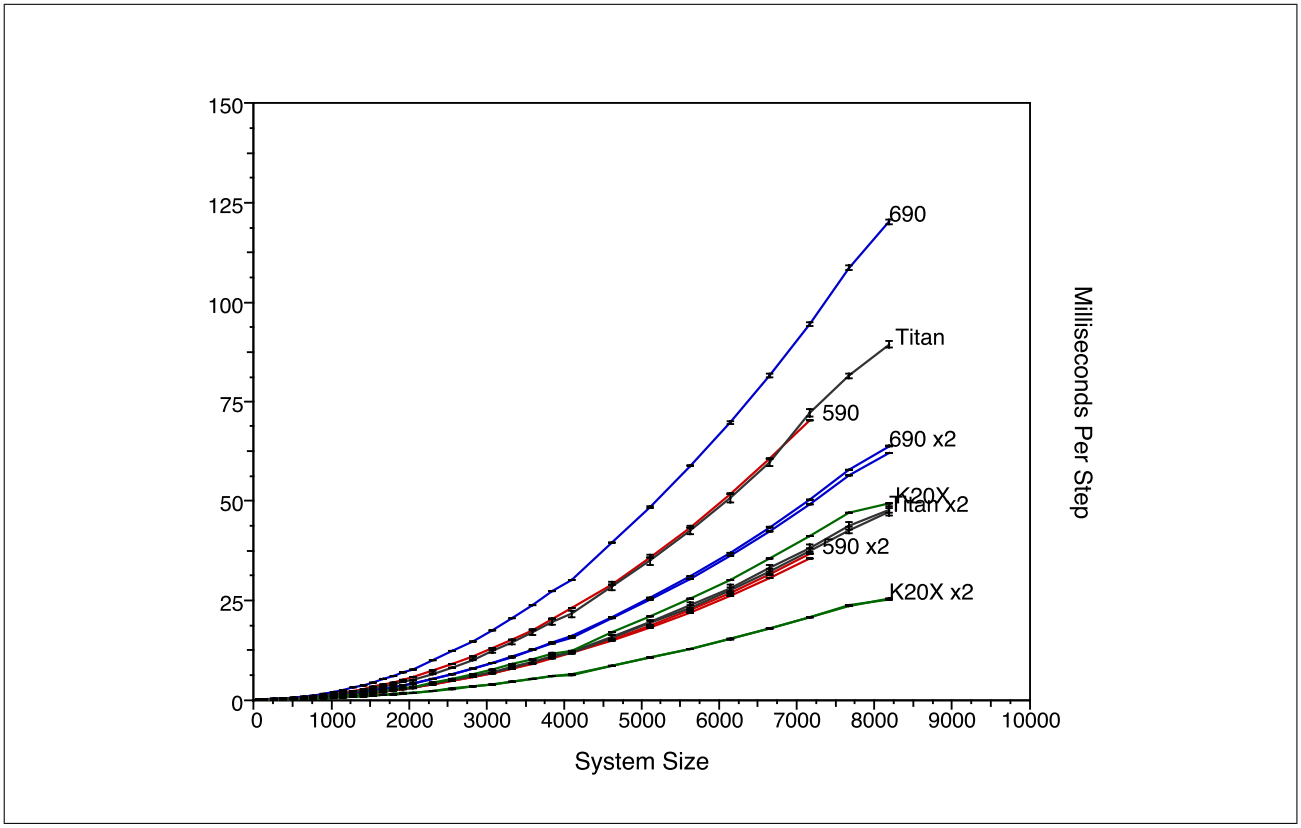


Figure 9: Performance results of the m-GPU shallow water simulations for system sizes from 64^2 to 8192^2 on four m-GPU systems - GTX590, GTX690, GTX780 and K20X.

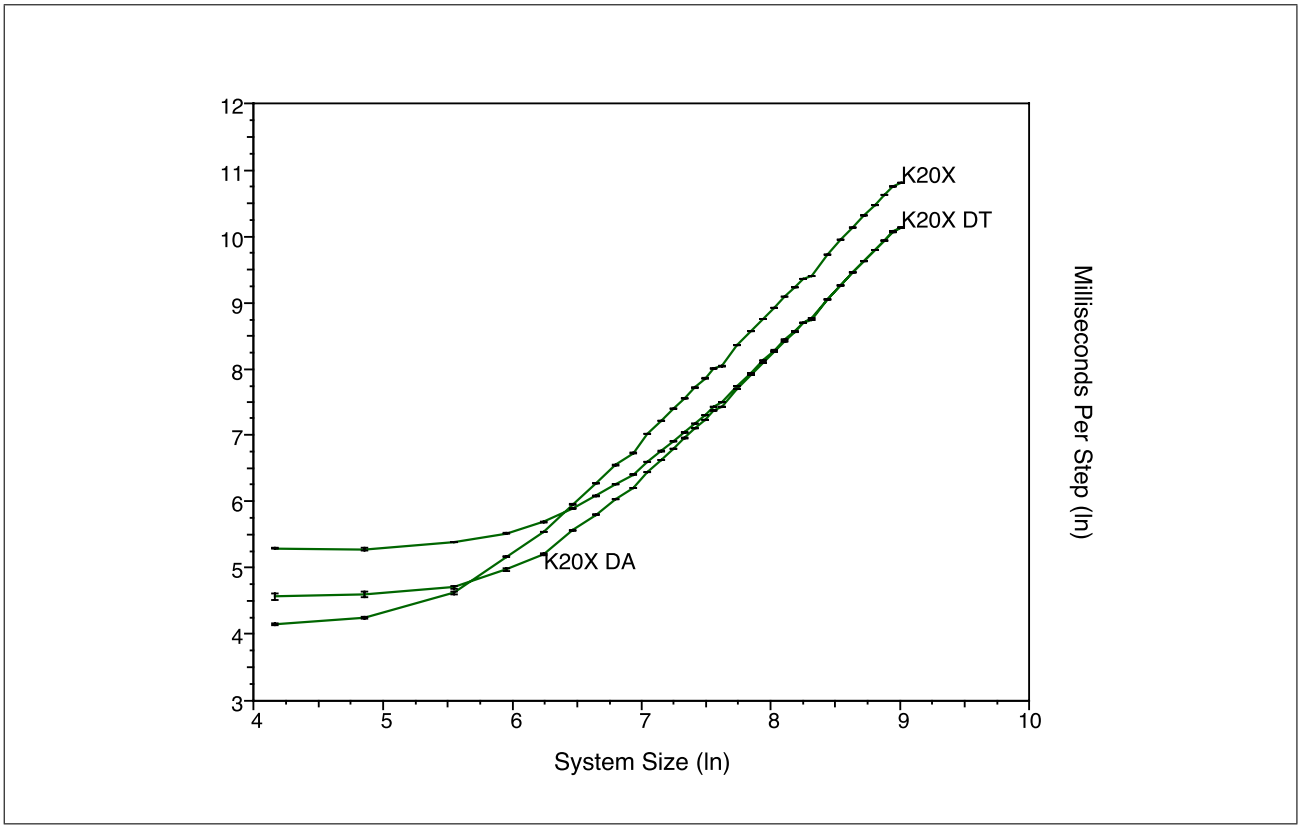


Figure 10: Performance results of the m-GPU shallow water simulations using Direct Access (DA) and Direct Transfer (DT) on 2x K20Xs compared with a single GPU simulation on a K20X. The results are shown in ln-ln scale.

munication. It is worth noting that the Tesla K20X and the GeForce GTX780 devices must communicate through the PCIe bus as the GPUs are in two separate cards while the GTX590 and GTX690 are both dual-GPU cards.

It can be seen from these results that (as is expected) the Tesla K20X compute cards easily provide the best performance. In fact the SW simulations running on a single K20X are only slightly slower than the same simulation running on two GTX780 devices.

One unusual result is the performance of the GTX590 devices which outperform the GTX690 and compete with the GTX Titan systems in both the single- and multi-GPU implementation. Given the nature of the floating-point based computation of the SW model it is not unexpected that the GTX590 may outperform the GTX690. The Fermi architecture GPUs have a significantly different architecture with more multiprocessors but only 32 cores per multiprocessor. In previous research it has been found that Fermi-architecture GPUs can outperform equivalence Kepler architecture GPUs for some floating-point processing problems (Johnson et al. 2013).

The performance results of the single-GPU and two m-GPU implementations of the SW model running on two K20X devices are shown in Figure 10. On this scale it is easier to see the difference between the implementations across the different system sizes. Similar relative performance is observed for all the m-GPU systems tested so only the results for the K20X are presented to make the graph easier to read.

Figure 10 show that the single-GPU implementation is the fastest for very small simulations (64^2 - 192^2) as these small systems cannot use the full computational power of a single-GPU let alone two. As the system sizes increases the speedup from using two GPU devices approaches the optimal 2x.

The Direct Access implementation provides the best performance of the two m-GPU implementations and provides a noticeable speedup over the Direct Transfer method from 2x for system size of 64^2 to 1.2x at 1024^2. For systems larger this the performance of the two implementations becomes almost indistinguishable but the Direct Access method never imposes a performance penalty.

Table 1: Current prices (estimated from 2013 online information) of the evaluated graphics cards and their relative performance.

Model	Price (NZD)	Speedup (vs 1x GTX590)
GTX590	$\approx \$1,200$	1x
GTX590 (2x GPU)	$\approx \$1,200$	1.92x
GTX690	$\approx \$1,400$	0.74x
GTX690 (2x GPU)	$\approx \$1,400$	1.40x
GTX Titan	$\approx \$1,800$	0.98x
GTX Titan x2	$\approx \$3,600$	1.86x
K20X	$\approx \$4,500$	1.71x
K20X x2	$\approx \$9,000$	3.38x

6 Discussion & Conclusions

We have described how a simulation of a shallow-water model can be implemented on a multi-GPU system using domain decomposition and two methods device-device communication. These implementations have been tested on several different GPU architectures to compare performance across recent devices.

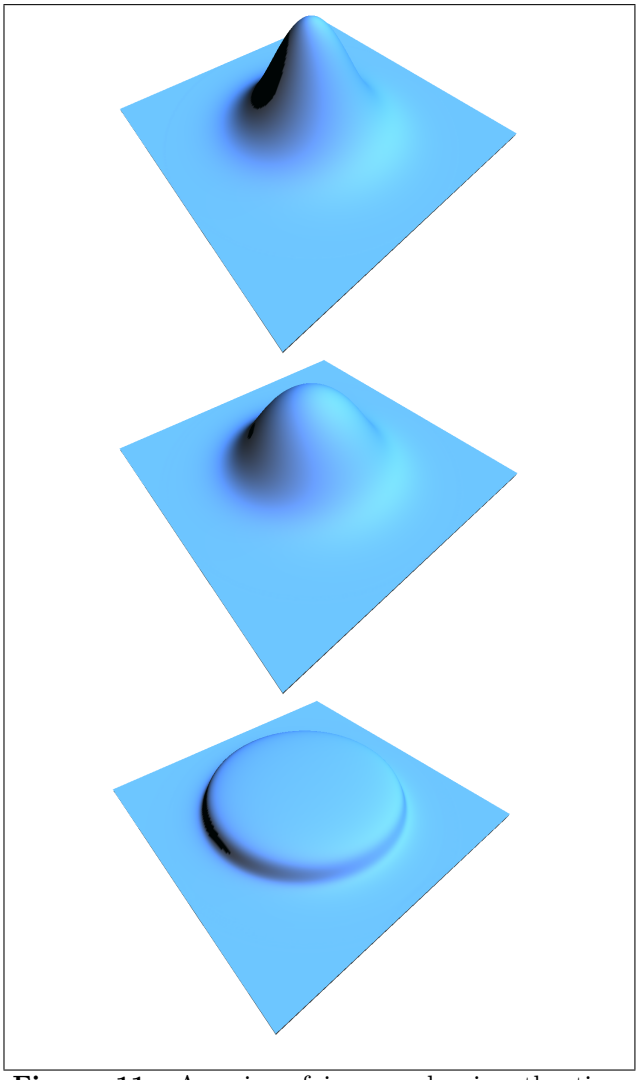


Figure 11: A series of images showing the time evolution of a shallow-water simulation computed on an m-GPU system. The system is initialised with a Gaussian curve at the centre.

The comparison of the different generation GPU devices has revealed the unexpected result that the older generation Fermi-architecture GTX590 can offer better performance than the newer Kepler-architecture equivalent card, the GTX690. Although the GTX690 contains many more cores than the GTX590, the configuration of the Kepler GK104 SMX units are not well suited to the memory and floating-point intensive kernels required by the shallow water model. This result was shown in both the single- and multi-GPU implementations. Given the relatively low price of the GTX 590 it shows that more expensive cards do not necessarily provide better CUDA performance.

Another surprising result of these experiments is the measured difference in performance between the GTX Titan and the K20X. Both of these cards should contain the same GK110 GPUs yet show a significant difference in performance. There is no clear reason for the performance difference as this simulation does not make use of the extra features enabled only in the K20X.

All the m-GPU capabilities devices tested show similar scaling behaviour with the Direct Access method providing a performance benefit for smaller

systems over the Direct Transfer method. The performance of these two implementation converges as the system size grows beyond 1024^2. At these larger system sizes the performance of both methods approaches the optimal 2x speedup over the single-GPU implementation.

Based on these findings we conclude that both methods are suitable for m-GPU implementations of this type of simulation using domain decomposition as they both offer similar performance for the large systems sizes m-GPU implementations would usually be used for. However, as the Direct Access method offers better performance for smaller systems and does not add significantly to the code complexity we find it to be the better method of device-device communication for this type of problem on both Fermi- and Kepler-architecture GPU devices.

Future work includes investigation of the performance difference between the GTX Titan and the K20X and an extension of this work to more GPU devices hosted in the same machine as well as GPU-accelerated compute clusters.

References

Bailey, C. L. (2010), Mathematical modelling of shallow water flows with application to Moreton Bay, Brisbane, PhD thesis, Loughborough University.

Bailey, D., Barscz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V. & Weeratunga, S. (1994), The nas parallel benchmarks, Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, CA, USA.

Barros, S. R. M. & Kauranne, T. (1991), On the parallelization of global spectral weather models. ECMWF preprint, submitted to Parallel Computing.

Barry, R. C. & Chorley, R. J. (1989), *Atmosphere, weather and climate*, 5 edn, Routledge.

C.A.J.Fletcher (1991*a*), *Computational Techniques for Fluid Dynamics*, Vol. 1, 2nd edn, Springer-Verlag. Fundamental and General Techniques.

C.A.J.Fletcher (1991*b*), *Computational Techniques for Fluid Dynamics*, Vol. 2, 2nd edn, Springer-Verlag. Specific Techniques for Different Flow Categories.

Dyson, G. (2012), *Turing's Cathedral: The Origins of the Digital Universe*, Vintage. ISBN: 978-1400075997.

Griebel, M., Dornseifer, T. & Neunhoeffer, T. (1998), *Numerical Simulation in Fluid Dynamics A Practical Introduction*, number ISBN 0-89871-398-6, SIAM.

Hawick, K. (2011), Visualising multi-phase lattice gas fluid layering simulations, *in* 'Proc. International Conference on Modeling, Simulation and Visualization Methods (MSV'11)', CSREA, Las Vegas, USA, pp. 3–9.

Hawick, K. A. (1991), Unified Weath/Climate Model Parallel Implementation Feasibility Study, In confidence, Edinburgh Parallel Computing Centre, Edinburgh University, EH9 3JZ, Scotland UK.

Hawick, K. A., Bell, R. S., Dickinson, A., Surry, P. D. & Wylie, B. J. N. (1992), Parallelisation of the unified model data assimilation scheme, *in* 'Proc. Fifth ECMWF Workshop on Use of Parallel Processors in Meteorology', European Centre for Medium Range Weather Forecasting (ECMWF), Reading.

Hawick, K. A., Bogucz, E. A., Degani, A. T., Fox, G. C. & Robinson, G. (1995), Computational Fluid Dynamics Algorithms in High-Performance Fortran, *in* 'Proc. AIAA 25th Computational Fluid Dynamics Conf'.

Hawick, K. A. & Playne, D. P. (2011), 'Hypercubic Storage Layout and Transforms in Arbitrary Dimensions using GPUs and CUDA', *Concurrency and Computation: Practice and Experience* **23**(10), 1027–1050.

Hurrell, J., Meehl, G. A., Bader, D., Delworth, T. L., Kirtman, B. & Wielicki, B. (2009), 'A unified modeling approach to climate system prediction', *Bull. Amer. Meteorological Soc.* **December**, 1819–1832.

Johnson, M. G. B., Playne, D. P. & Hawick, K. A. (2010), Data-parallelism and gpus for lattice gas fluid simulations, *in* 'Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'10)', CSREA, Las Vegas, USA, pp. 210–216. PDP4521.

Johnson, M. G. B., Playne, D. P. & Hawick, K. A. (2013), Performance tradeoff spectrum of integer and floating point applications kernels on various gpus, *in* 'Proc. 13th International Conference on Computer Design (CDES'13)', number CSTN-180, WorldComp, p. CDE4086.

Leist, A., Playne, D. P. & Hawick, K. A. (2009), 'Exploiting Graphical Processing Units for Data-Parallel Scientific Applications', *Concurrency and Computation: Practice and Experience* **21**(18), 2400–2437. CSTN-065.

Lyes, T. S., Johnson, M. G. B. & Hawick, K. A. (2012), Visual simulation of a multi-species coloured lattice gas model, *in* 'Proc. Int. Conf. on Scientific Computing (CSC'12)', CSREA, Las Vegas, USA, pp. 115–124.

Martinez, C. J., Campbell, K. L., Annable, M. D. & Kiker, G. A. (2008), 'An object-oriented hydrologic model for humid, shallow water-table environments', *Journal of Hydrology* **351**, 368–381.

Micikevicius, P. (2009), 3D finite difference computation on GPUs using CUDA, *in* 'GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units', ACM, New York, NY, USA, pp. 79–84.

Muller, M., Stam, J., James, D. & Thurey, N. (2008), Real time physics class notes, Technical report, NVIDIA. Chapter 11 - Shallow Water Equations.

NVI (2012), *CUDA C Programming Guide*, 5.0 edn.

Peixato, J. P. & H.Oort, A. (1984), 'Physics of Climate', *Rev.Mod.Phys.* **56**(3), 365–429.

Playne, D. P. & Hawick, K. A. (2011), 'Comparison of GPU Architectures for Asynchronous Communication with Finite-Differencing Applications', *Concurrency and Computation: Practice and Experience (CCPE)* **Online**, 1–11. **URL:** *http://onlinelibrary.wiley.com/doi*

Randall, D. A. (2006), The shallow water equations, Technical report, Atmospheric Science, Colorado State University, Fort Collins, Colorado, USA.

Schmidt, J., Berzins, M., Thornock, J., Saad, T. & Sutherland, J. (2013), Large scale parallel solution of incompressible flow problems using uintah and hypre, *in* 'IEEE International Symposium on Cluster Computing and the Grid', IEEE Computer Society, Los Alamitos, CA, USA, pp. 458–465. ISBN 978-1-4673-6465-2.

Tett, S. F. (1991), A massively parallel algorithm for the spectral method. UKMO Preprint.

Tritton, D. (1988), *Physical Fluid Dynamics*, 2 edn, Clarendon Press.

Vinas, M., Lobeiras, J., Fraguela, B. B., Arenaz, M., Amor, M., Garcia, J. A., Castro, M. J. & Doallo, R. (2013), 'A multi-gpu shallow-water simulation with transport of contaminants', *Concurrency and Computation: Practice and Experience* **25**, 1153–1169.

Vu, V. T., Cats, G. & Wolters, L. (2008), Asynchronous Communication in the HIRLAM Weather Forecast Model, *in* 'Proc. 14th Annual Conference of the Advanced School for Computing and Imaging (ASCI)'.

W.D.McComb (1990), *The Physics of Fluid Turbulence*, Clarendon Press.

Xiao, Q., Zou, X., Pndeca, M., Shapiro, M. & Velden, C. (2002), 'Impact of gms-5 and goes-9 satellite-derived winds on the prediction of a norpex extratropical cyclone', *Monthly Weather Review* **130**(3), 507–528. **URL:** *http://journals.ametsoc.org/toc/mwre/130/3*

Zaspel, P. & Griebel, M. (2012), 'Solving incompressible two-phase flows on multi-gpu clusters', *Computers & Fluids* **In press**, 1–9.

3D FPGA versus Multiple FPGA System: Enhanced Parallelism in Smaller Area

[*]Krishna Chaitanya Nunna, [†]Farhad Mehdipour and [♀]Kazuaki Murakami

[*♀]Department of Advanced Informatics
[†]E-JUST Center
Graduate School of Information Science and Electrical Engineering
Kyushu University
Fukuoka, Japan

[*]krishna@soc.ait.kyushu-u.ac.jp, [†]farhad@ejust.kyushu-u.ac.jp, [♀]murakami@ait.kyushu-u.ac.jp

Abstract

Handling large amounts of data is being limited by bandwidth constraint between processors components and their memory counterparts. Three-dimensional integration (3D) is providing possible solution to handle such critical applications. Especially for running larger designs when implementing on multi-FPGA system, which can produce huge amount of fine-grain parallelism, for satisfying the speed and reliability needs, 3D FPGA can be considered as a close candidate to choose. In this paper we tried to show the benefits of running larger designs on 3D FPGA compared to running on multi-FPGA systems using benchmark simulation. Results showed that a TSV-based 3D FPGA achieved better performance and area results.

Keywords: 3D FPGA, TSV, Multi-FPGA System

1 Introduction

One of the vital applications of field-programmable gate arrays (FPGAs) is rapid application specific integrated circuit (ASIC) prototyping. An FPGA is an array of programmable logic blocks (LBs), configurable interconnect, and I/O blocks which can be user-configured to implement complex digital circuits. This highly reprogrammable structure enables FPGAs to exploit parallelism at different granularities. ASIC designers are turning to FPGA for prototyping their designs to take the advances of their low cost and fast prototyping. Especially for simulations and design verification of larger circuits multiple FPGA systems became common. A multi-FPGA system, as shown in Fig. 1(a) contains multiple reprogrammable devices on a PCB. A system of FPGAs can be seen as a computing substrate with somewhat different properties than standard microprocessors. It provides a huge amount of fine-grain parallelism. For example, a parallel sparse matrix solution for direct circuit simulation on multi-FPGA system is presented by Nechma (2012). The re-programmability of the FPGAs allows one to download algorithms onto the FPGAs, and change these algorithms just as PC can change programs. When a circuit is needed to implement on to a multi-FPGA system, it is partitioned into number of parts equal to number of FPGA chips on the system. Then these partitions will be mapped on those FPGAs separately. The communication between the chips is performed by inter-chip connects. Note that the bus shown in Fig 1(a) is an example representation of such communication. In real, the way the FPGAs connected depends totally on type of chip package used. For example SPP-2K reconfigurable platform shown in HitechGlobal (2012) having nine Xilinx Virtex-4 FPGAs along with external interface can be used for ASIC and IP development. Even though multi-FPGA system is able to handle larger designs, due to their off-chip communication strategy the communication between the chips is limited by the bandwidth constraints from the interface side. With the constraint as limited number of I/O pads on FPGA, it is also necessary to multiplex the FPGA-to-FPGA signals, which further reduces the performance. One possible solution to achieve higher speed at the same level of circuit complexity is three-dimensional (3D) integration of FPGAs.

3D FPGA is one of the promising innovations which can provide benefits like increasing transistor density, reduced form factor, heterogeneous architectures and improvement in delay by significantly reducing the wire lengths of integrated circuits (J. Alexander *et al*. 1996). It is a multi-layer device stacked using through-silicon via (TSV) technology. That means the communication between the layers is done by using TSVs as shown in Fig. 1(b). The communication between the layers in 3D FPGA is on-chip, and hence it is quite obvious from the implementation perspective to expect higher speed compared to the off-chip communication platform. An interesting work is presented by Zied *et al*. (2012) on performance comparison between multi-FPGA system and a Xilinx 2.5D FPGA. In their experiments they compared two ways of implementing specific ASIC designs. The first used the DN2076K10

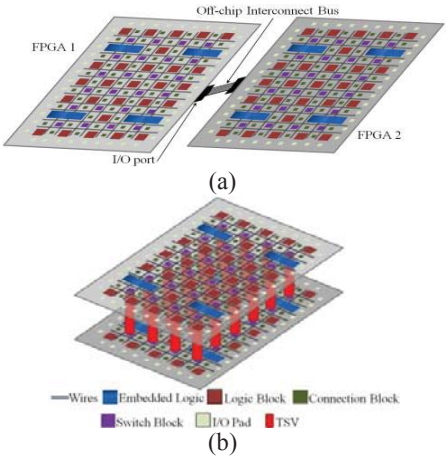


Figure 1: (a) Multi-FPGA system using off-chip interconnect bus (b) TSV-based 3D FPGA

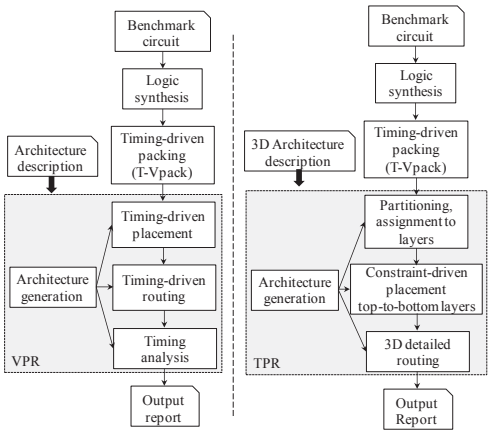


Figure 2: (a) VPR CAD flow for 2D FPGAs (Betz *et al.* 1997) (b) TPR CAD flow for 3D FPGAs (Ababei *et al.* 2006)

DINI board (DiniGroup 2013), which contains six FPGA Virtex-6 LX760 FPGAs (each one has 1 die/chip), with a total logic capacity of around 2.8 million LUTs. The second used a board (XCVirtex7 2013) designed with two Virtex-7 2000T FPGAs (each one has 4 dies/chip), with a total logic capacity of around 4 million LUTs and a maximum logic utilization of 70%. Virtex-6 is a 2D FPGA where as Virtex-7 is 2.5D FPGA in which multiple FPGA dies in each package are linked using stacked silicon interconnect layer. They have showed that the clock frequency is increased in the case of 2.5D FPGA against multi-FPGA system. Even though they tried to show the benefits pertained to 2.5D FPGA, Virtex-7 is not a fully 3D which is having similar footprint area as multi-FPGA system and hence more than half footprint reduction that can be achieved in real 3D FPGA is not visible in their experimental results. For example, Davis, *et al*. (2001) reported a 3x reduction in total silicon area and a 12x reduction in chip footprint for a monolithic 3D IC with 4 device layers when compared to a 2D IC. The reduction in wire lengths enabled a size decrease of the logic gate drivers for these wires, which reduced the distance between logic gates further, causing a significant reduction in silicon area. Earlier works introduced a novel partitioning techniques for multi-FPGA systems (Swaminathan *et al*. 2012, Kim *et al*. 1995 and Roy *et al*. 1995) and power-aware partitioning techniques and thermal-aware placement and routing techniques for 3D FPGAs (Krishna *et al*. 2013 and Siozios *et al*. 2010). Most of the earlier research work, except Zied *et al*. (2012) which showcased the benefits of 2.5D FPGA compared to a multi-FPGA system, concentrated on benefits attained by multi-FPGA system and different techniques to improve the performance metrics for the system.

This research work presents emphatic analysis on benefits attained by TSV-based stacked 3D FPGA against the multi-FPGA system specifically for rapid prototyping application. The key contributions are: performance evaluation of 3D FPGA and multi-FPGA system against 14 largest MCNC benchmark circuits; performance and area comparison. For a fair comparison between multi-FPGA system and 3D FPGA, real time board-based experimental environment is not available either in academia or domestic market. It is quite reasonable to show the benefits of 3D FPGA as an alternative through potential CAD tool environment which can cope with design and development of 3D integration. Up to our knowledge our work presented in this paper is first of its kind.

The remainder of the paper is followed by background in section 2, methodology for proposed work in section 3 and finally concluded along with results in section 4.

2 Background

As a part of the analysis on MCNC benchmark circuits, we are using Versatile Place and Route (VPR) (Betz *et al*. 1997) and Three-dimensional Place and Route (TPR) (Ababei *et al*. 2006) which are widely used research and academia placement and routing tools for 2D and 3D FPGAs respectively. VPR is an open-source place and route tool intended to research in CAD and architecture for island-style FPGA architectures. These FPGAs contain I/O blocks and logic blocks surrounded by programmable routing. As the logic blocks are all assumed to be identical, a single logic block and its adjacent routing can be combined to form a tile that can be replicated to create the full FPGA. The experimental process used in VPR-based CAD and architecture research is illustrated in Fig. 2(a). This flow takes an input circuit at the logic level and FPGA architecture along with necessary design specifications and implements the circuit on the specified FPGA. The output generates all the required parameters starting from the number of interconnects occupied by the nets in circuit to the critical path delay of the implemented circuit. It can also model an approximation of the area taken up by that circuit in that FPGA. In the first step of the flow, the circuit is synthesized and technology mapped. The output after technology mapping must then be packed into the logic clusters available on the FPGA. Once packing is complete, VPR is then used to perform placement and routing for the

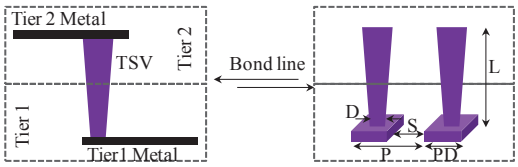


Figure 3: (a) Typical TSV representation fabricated on a 2-layer stacked IC (b) Typical TSV parameters representation (Papanikolaou *et al.* 2011)

Parameter	2008	2009	2010	2011	2012	2013	2014	2015
TSV diameter, D (um)	1.6	1.5	1.4	1.3	1.3	1.2	1.2	1
TSV pitch, P (um)	5.6	5.5	4.4	3.8	3.8	2.7	2.6	2.5
Pad spacing, S (um)	1	1	1	0.5	0.5	0.5	0.5	0.5
Pad diameter, PD (um)	4.6	4.5	3.4	3.3	3.3	2.2	2.1	2

Table 1: TSV parameters reported by 2008 ITRS data

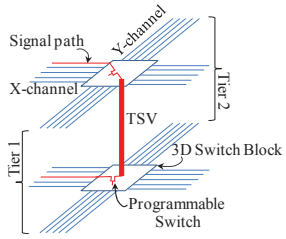


Figure 4: Net connected between two layers using TSV in association with programmable switch

circuit. Finally, timing analysis is completed to determine the performance of the circuit on the FPGA.

TPR is a partitioning-based placement and routing toolset. Its purpose is to serve the research community in predicting and exploring potential gains that the 3-D technologies for FPGAs have to offer (similar to the role VPR played in the development of FPGA physical-design algorithms). It can be used as a platform for development and implementation of new ideas in placement and routing for 3-D FPGAs. The philosophy of TPR closely follows that of its 2-D counterpart, VPR. The flow of the TPR based placement and routing CAD tool is shown in Fig. 2(b). The design flow starts with a technology mapped netlist in .blif format. The .blif netlist is converted to a netlist composed of more complex logic blocks. The .net netlist as well as the architecture description file are inputs to the placement algorithm. The placement algorithm first partitions the circuit into a number of balanced partitions equal to the number of layers for 3-D integration. The goal of this first min-cut partitioning is to minimize the connections between layers, which translates into minimum number of vertical (i.e., interlayer) wires. After dividing the netlist into layers, TPR continues with the placement of each layer in a top-down fashion.

Benefited from the stacking of layers, the total interconnect length of 3D chip is expected to be shorter than that of 2D chip so that the footprint area of 3D IC becomes smaller than that of 2D IC. Wirelength reduction in 3D ICs has been demonstrated in earlier studies including real chip design (Joyner *et al.* 2000, Kim *et al.* 2009 and Thorolfsson *et al.* 2009). Even though the total wirelength is reduced, the delay associated with the TSV is affected by its large capacitance which one should consider in delay calculations. For example, 20fF TSV capacitance is comparable to the capacitance of a 120µm-long intermediate-layer wire (e.g. M4 to M6) in 45nm technology (Kim *et al.* 2010). In addition, TSV RC has different characteristics from wire RC. Therefore, it is shown that if we use wire RC models for TSV parasitics, there will be non-negligible amount of errors in computation and prediction of TSV-related delay and power consumption in 3D chips. A TSV connected between two tiers in a 2-layer 3D device is shown in Fig. 3(a). The width or diameter of a TSV is larger than a normal metal interconnect which we can find on a single die. Typical TSV parameters are represented as shown in Fig. 3(b) and their predicted values by ITRS are shown in Table 1. The delay pertained to TSV can be calculated using Elmore delay model as:

$$T_{TSV} = 0.5 * R_{TSV} * C_{TSV} * L_{TSV} + R_{swt} * C_{metal} + T_{swt} \quad (1)$$

The above equation represents the delay associated with TSV in combination with the switch connected to it as shown in Fig. 4. In Eq. (1), R_{TSV} is resistance of TSV, C_{TSV} is capacitance of TSV, L_{TSV} is length of TSV, R_{swt} is the resistance of switch, C_{metal} is capacitance of the metal, T_{swt} is switch delay associated with TSV.

3 Methodology

The proposed methodology on performance evaluation of 3D FPGA and multi-FPGA system is specific to logic simulation and uses both VPR and TPR to generate necessary metrics and parameters. Here the partitioning technique is the significant step for knowing the number of connections between layers (or chips) by partitioning the input circuit equal to the number of layers (or chips), which we assumed as two in our experiments. The partitioning tool used by TPR is hMetis presented by Karypis *et al.* (1997) which is a closed-source tool. In order to have better control on the partition process we have replaced hMetis with ParKway presented by Trifunovic *et al.* (2004) which is an open-source tool. Since the Parkway is a heuristic based approach with undeterministic result, we performed partitioned more than two times and the best result among these is selected.

As a part of the off-chip interconnect delay calculation for a multi-FPGA system, we extract the required information such as cut size from the partitioned circuit performed by ParKway. By utilizing the extracted parameters, total delay including off-chip interconnect delay for every signal that traverses between two FPGAs is modelled as similar to the work proposed by Swaminathan *et al.* (2012).

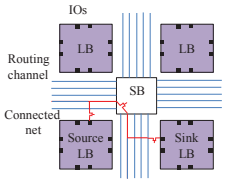


Figure 5: LB-LB connection for delay calculation

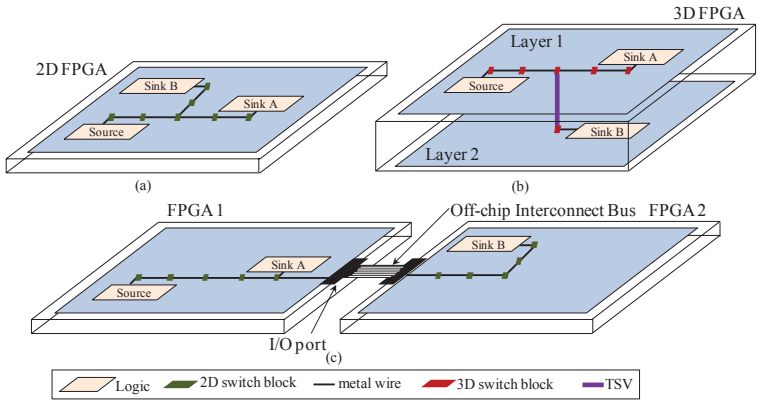


Figure 6: (a) Circuit placed on 2D FPGA (b) Partitioned circuit placed on a TSV based 2-layer stacked 3D FPGA (c) Partitioned circuit placed on multi-FPGA system with two FPGAs connected using an off-chip interconnected bus.

$$T_{total,multi-FPGA} = \lceil \frac{N_{cuts}}{N_{pin\_avail}} \rceil * T_{clk} + (T_{wire\_max} * d * N_{arch}) \quad (2)$$

The first part of Eq. (2) represents the delay associated with multiplexing of the signals through off-chip interconnects bus. As in some cases, the number of off-chip nets (or cut nets) may be more than the available number of pins in each FPGA of multi-FPGA system. For handling such critical situations, multiplexing of signals is required which may contribute majority of the time delay as proved in the following sections. In the first part of Eq. (2), N_{cuts} is the number of cuts between two partitions, N_{pin_avail} is the number of pins available to each partition. This can be calculated based on the chip specifications. For the example FPGA in our experiments, each pin on the four sides of FPGA shares two routing channels (X-channel/Y-channel). That means if an FPGA has 20 routing channels on either side of the chip, then the total number of pins available is (20/2) * number of sides of FPGA chip. Hence for our experiments, this N_{pin_avail} will change for each benchmark circuit based on the minimum required array size of FPGA. T_{clk} is the critical path delay obtained when the input circuit is actually implemented on a single 2D FPGA using VPR (to decide a maximum clock frequency that a given circuit can operate). This is to ensure to find the best timing results such that the maximum operating frequency will be decided based on VPR results.

In the second part of Eq. (2), T_{wire_max} is the maximum wire delay that can be obtained for a given FPGA array, d is the maximum distance that the signal can travel from a certain point on FPGA array to the edge of the chip surface (to connect IO pad) and N_{arch} is the number of FPGAs used in the multi-FPGA system which 2 in our experiments. Here T_{wire_max} can be obtained by finding maximum wirelength across a given FPGA area as a multiplication of number of LBs in a row of LBs on a square shaped FPGA with the delay between any two LBs. The delay between any two LBs can be calculated as shown in Fig. 5. The connected path shown in red colour represents the communication between any two neighbouring LBs. It is clear from the figure that the delay calculated between these neighbours involves three switches associated with SB and two IO connectors to the routing channel.

The above explanation about delay calculation for second part of Eq. (2) is a general case where the original delay pertained to each FPGA in a multi-FPGA system is not known. Since we are representing the real time multi-FPGA system using TPR, the delay calculated for that second part of Eq. (2) is taken originally from TPR as represented in Eq. (3) and (4). Measuring the critical path in 3D FPGA is easy as it can be directly calculated by TPR itself. For multi-FPGA system case, we are using the same delay information obtained by TPR with the following approach. We first separate the total delay, obtained from TPR, into three parts: layer 1 delay, TSV delay, layer 2 delay as shown in Eq. 3.

$$T_{total,3D\ FPGA} = T_{layer1} + T_{TSV} + T_{layer2} \quad (3)$$

$$T_{total,multi-FPGA} = \lceil \frac{N_{cuts}}{N_{pin\_avail}} \rceil * T_{clk} + (T_{layer1} + T_{layer2}) \quad (4)$$

These separated delay variables in Eq. (3) can be obtained by parsing the TPR source code and keeping the watch-dogs to read the information at the necessary points. Now the TSV delay, T_{TSV} is replaced by the first part of the Eq. (2) as shown in Eq. (4). The total (T_{layer1} + T_{layer2}) represents the delay associated with two FPGA chips in the given multi-FPGA system. In TPR case, the goal of placement and routing is to reduce the wirelength as much as they can. That means any connection inside each layer will be made by utilizing the nearest TSV in such a way that the logic blocks placed in opposite layers will be connected with shorter wirelength. For multi-FPGA system, the same phenomenon can be applicable. In a system of multiple FPGAs, each chip will be occupied in such a way that the logic that needs to be connected through off-chip bus will be placed near to the I/O pins. This is helpful in minimizing the wirelength in each FPGA by satisfying the area constraint. Fig. 6 represents the circuit placed and routed on (a) a 2D FPGA, (b) 2-layer 3D FPGA and (c) multi-FPGA system with two FPGAs connected using an off-chip interconnected bus.

4 Experiment Results

We have performed all experiments on Ubuntu Linux machine. The assumptions we use for all the simulations are

Category	Parameter	Value
Technology		45nm
Architecture	Shape	Square
	Style	Island
	LUT size	4
	Number of layers in 3D FPGA	2
	Number of FPGAs in multi-FPGA system	2
	Each FPGA array size	Best fitted to the circuit size
	Routing channel width (X and Y directions)	Best fitted to the circuit size
TSV	shape	Square
	type	Via-first
	die-bonding type	Face-to-back
	width	2.47μm
	resistance	70 Ω
Local and Global Interconnects	As per the NCSU's FreePDK45nm (FreePDK 2013)	

Table 2: Parameters assumed for experiments

shown in Table 2. As part of the experiments the steps involved are:

- Implement the input circuit on TPR (3D FPGA) and VPR (2D FPGA) to find the best fit FPGA array size. Thus gives: array size for each layer; number of IOs for each layer; number of inter-communication nets; chip area in terms of wirelength, circuit delay and TSV delay.
- Calculate the total delay including off-chip interconnect delay for multi-FPGA system using Eq. 4 for the input circuits based on the parameters obtained in step 1.

4.1 Effective area utilization

We first implemented all MCNC benchmark circuits on 2-layer 3D FPGA using TPR to get the best fit array size in terms LBs (row x column). For better understanding to the reader we are also showing the best fit FPGA array sizes when implemented on 2D FPGA using VPR. All the 14 MCNC benchmark circuits are implemented on these 2D and 3D FPGAs such that the overall chip utilization will be more than 90% as shown in Table 3. This is because of the reason that the fundamental application of multi-FPGA is implementing larger circuits. That means each FPGA in a given multi-FPGA system will occupied almost fully by the input circuit. Hence during our experiments we make ensure to achieve more than 90% area utilization on each layer of the 3D FPGA. For a fair comparison on advantages of 3D FPGA over its counterpart, wirelength and routing area numbers obtained for 2D and 3D FPGAs are explained in the following sections. As a target work, the critical path delay obtained only for 3D FPGA is compared against the delay obtained for multi-FPGA system.

Benchmark	LBs	Nets	Total nets including multi-terminal nets	Input pins	Output pins	2D FPGA			3D FPGA		
						Routing Channel Width	LBxLB	% Utilization	Routing Channel Width	LBxLB	% Utilization
alu4	1522	1536	5408	14	8	19	40x40	95.13	18	29x29x2	90.49
apex	1878	1916	6692	38	3	21	44x44	97.00	20	32x32x2	91.70
bigkey	1707	1936	6313	300	197	13	42x42	96.77	18	30x30x2	94.83
clma	8383	8445	30462	63	82	24	92x92	99.04	28	66x66x2	96.22
des	1591	1847	6110	256	245	16	41x41	94.65	19	29x29x2	94.59
diffeq	1497	1561	5296	65	39	15	39x39	98.42	19	28x28x2	95.47
elliptic	3604	3735	12634	132	114	20	61x61	96.86	19	43x43x2	97.46
frisc	3556	3576	12772	21	116	23	60x60	98.78	23	43x43x2	96.16
pdc	4575	4591	17193	16	40	32	68x68	98.94	39	49x49x2	95.27
s298	1931	1935	6951	5	6	16	44x44	99.74	20	32x32x2	94.29
s38417	6406	6435	21344	30	106	18	81x81	97.64	31	57x57x2	98.58
s38584.1	6447	6485	20840	39	304	18	81x81	98.26	19	58x58x2	95.82
seq	1750	1791	6193	41	35	22	42x42	99.21	25	30x30x2	97.22
spla	3690	3706	13808	16	46	24	61x61	99.17	38	45x45x2	91.11
Average	3467	3535	12287	74	96	20	60x60	98	24	42x42x2	95

Table 3: FPGA chip area utilization rates for 2D FPGA and a 2-layer 3D FPGA

Circuit	Total Wirelength for 2D FPGA (1 LB units)	Total Wirelength for 3D FPGA (1 LB units)	% Wirelength Reduction
alu4	40854	37300	8.70
apex	63646	55552	12.72
bigkey	44622	37826	15.23
clma	332187	270975	18.43
des	55068	52549	4.57
diffeq	41740	30602	26.68
elliptic	105066	90937	13.45
frisc	120553	111641	7.39
pdc	266472	194067	27.17
s298	56507	42184	25.35
s38417	259953	128930	50.40
s38584.1	160413	111520	30.48
seq	61979	53340	13.94
spla	191049	121450	36.43
Average	128579	95634	20.78

Table 4: Wirelengths achieved for 2D FPGA and a 2-layer 3D FPGA

Circuit	Routing Area for 2D FPGA (minimum width transistor)	Routing Area for 3D FPGA (minimum width transistor)	% Routing Area Reduction
alu4	4998700	3113250	37.72
apex	5001470	3998590	20.05
bigkey	3826720	3366830	12.02
clma	25386200	19456300	23.36
des	5991330	4862240	18.85
diffeq	3771640	2236380	40.71
elliptic	8192200	7377550	9.94
frisc	9450570	8126710	14.01
pdc	20216600	13980600	30.85
s298	4707840	2955570	37.22
s38417	20312900	11285200	44.44
s38584.1	15836200	11285200	28.74
seq	4886080	3789430	22.44
spla	15753100	8892540	43.55
Average	10595111	7480456	27.42

Table 5: Routing areas calculated for 2D FPGA and 2-layer 3D FPGA

4.2 Footprint area comparison

Table 4 shows the total wirelength required for realizing each circuit on a 2D FPGA and 3D FPGA. Table 5 shows the routing area required by 2D FPGA and 3D FPGA in terms of minimum width transistor. It is clear from the table that the 3D FPGA is effective in reducing the wirelength and routing area by an average of 20.78% and 27.42% respectively for the assumed MCNC benchmark circuits. This comparison is to show the effectiveness of wirelength reduction and hence the footprint reduction due to 3D integration of individual 2D layers. Note that the table doesn't represent the multi-FPGA system as the footprint area of the system is a simple multiplication of each layer area in 3D FPGA by number of layers. That means for a multi-FPGA system of having two FPGAs can have a footprint area larger than a 2-layer 3D FPGA plus additional off-chip interconnect bus area.

4.3 Performance comparison

The circuit delay involving off-chip communication delay for 14 MCNC benchmark circuits implemented on a multi-FPGA system with two FPGAs is shown in Table 6. In the table, number of off-chip nets (or cut nets) is obtained from partitioning each input circuit. Total number of available pins is obtained by using the architecture specification (number X- and Y- routing channels, shape of chip). The clock speed is obtained by running the input circuit for one time using VPR. The delay for each FPGA in the assumed multi-FPGA system is calculated based on Eq. (3) and (4). It is clear from the Table 6 that the multiplexing of signals contributes majority of the time delay. As shown in the Table 7, 3D FPGA achieved more than 80% delay saving compared to a multi-FPGA system of two FPGAs. For the benefit of user understanding, table also shows the delays for a 2D FPGA. Fig. 7 represents the comparison between 3D FPGA and multi-FPGA system of two FPGAs. It is observed that the majority of excess delay in multi-FPGA system is due to the multiplexing strategy. This is directly related to the IO pin constraint. But in the case of 3D FPGA, because of the vast number of TSVs, this constraint may not be seen. For example, a 30 x 30 FPGA array with a

Benchmark	Multi-FPGA		Number of off-chip nets (N_{cuts})	Total no. of available pins (N_{pin_avail})	Clock speed (T_{clk}) (ns)	Delay for Multiplexing	Delay for both FPGAs (T_{layer1}+ T_{layer2})	Total Delay (ns)
	LB Array	% Utilization						
alu4	29x29x2	90.49	105	58	8.99	32.53	4.98	37.51
apex2	32x32x2	91.70	103	64	13.52	43.51	4.77	48.28
bigkey	30x30x2	94.83	12	60	9.00	3.60	2.96	6.56
clma	66x66x2	96.22	175	134	35.06	91.58	18.53	110.11
des	29x29x2	94.59	67	58	9.90	22.87	4.63	27.50
diffeq	28x28x2	95.47	62	56	9.92	21.96	4.03	25.99
elliptic	43x43x2	97.46	109	88	20.62	51.09	7.38	58.48
frisc	43x43x2	96.16	180	86	19.52	81.70	6.48	88.18
pdc	49x49x2	95.27	338	98	22.36	154.26	10.39	164.65
s298	32x32x2	94.29	39	64	25.39	30.95	7.89	38.84
s38417	57x57x2	98.58	48	116	20.43	16.91	5.64	22.55
s38584	58x58x2	95.82	31	116	17.55	9.38	5.01	14.39
seq	30x30x2	97.22	152	62	10.96	53.72	6.22	59.94
spla	45x45x2	91.11	214	88	19.43	94.50	10.49	104.99
Average:	42x42x2	94.94	117	82	17.33	50.61	7.10	57.71

Table 6: Delay calculated using Eq. (4) for a multi-FPGA system having two FPGAs with off-chip interconnect bus

Circuit	Delay for 2D FPGA (ns)	Delay for 3D FPGA (ns)	Delay for Multi-FPGA (ns)
alu4	8.99	6.64	37.51
apex	13.52	7.12	48.28
bigkey	9.00	4.16	6.56
clma	35.06	25.04	110.11
des	9.90	6.52	27.50
diffeq	9.92	5.67	25.99
elliptic	20.62	9.98	58.48
frisc	19.52	8.64	88.18
pdc	22.36	14.04	164.65
s298	25.39	11.12	38.84
s38417	20.43	8.05	22.55
s38584.1	17.55	7.37	14.39
seq	10.96	8.64	59.94
spla	19.43	14.17	104.99
Average	17.33	9.80	57.71

Table 7: Delays obtained for 2D FPGA and 2-layer 3D FPGA

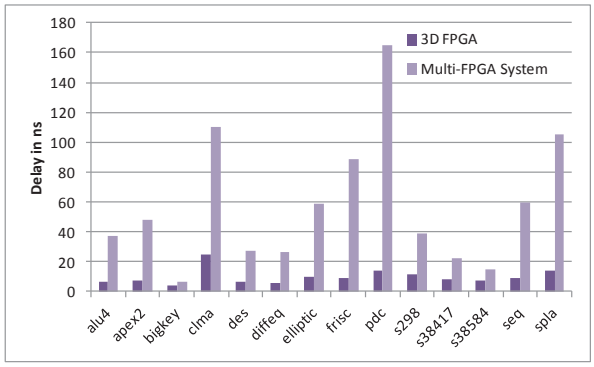


Figure 7: Delay comparison: 3D FPGA vs Multi-FPGA system

vertical routing channel width of 6 (6 TSVs connected to each switch block) has nearly 900 switch blocks and each switch block is connected to at least one TSV. That means the given array will have nearly 5400 TSVs (900 switch blocks x 6 TSVs/switch block) which are more than sufficient to accommodate all the cut nets between the layers. Literally, multiplexing of signals is not necessary in 3D FPGA which results in vast reduction in the delay. The reduced delays for all the 14 MCNC benchmark circuits implemented on a 2-layer 3D FPGA are depicted in Fig. 7.

5 Conclusion

In this paper, we have investigated the delay characteristics of multi-FPGA system and TSV-based 3D FPGA. We have showed that the delay improvement achieved by 3D FPGA is on par compared to multi-FPGA system majorly due its on-chip communication facility via TSVs. This kind of motivation can add extra potential to the already available feature of fine-grain parallelism of FPGAs which may results in much faster functional modelling. For the experiments we have consider square-shaped TSV with parameters similar to industry standards. We believe that the in-detail modeling of TSV along with its effect on the

material around it could help in showing added benefits in terms of the performance. As a part of the future work, heterogeneous FPGA architectures with more than two layers can be considered for targeting design flexibility towards performance improvements.

6 References

Tarek Nechma (2012): Parallel Sparse Matrix Solution for Direct Circuit Simulation on a Multiple FPGA System. Ph.D. thesis. University of Southampton.

HitechGlobal (2012):

http://hitechglobal.com/Boards/MultiFPGA.htm

J. Alexander, J. P. Cohoon, J. L. Colflesh, J. Karro, E. L. Peters, and G. Robins (1996): Placement and Routing for Three-Dimensional FPGAs. *Proc. 4th Canadian Workshop Field-Programmable Devices*, pp. 11–18.

Zied Marrakchi, Ramsis Farhat and Ramine Roane (2012): Improving ASIC prototyping on multiple FPGAs through better Partitioning. *An article in Tech Design Forum*.

DiniGroup (2013):

http://www.dinigroup.com/new/DN2076k10.php

XCVirtex7 (2013): http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm

J. Davis, et al. (2001): Interconnect limits on gigascale integration (GSI) in the 21st century. *Proc. IEEE*, vol.89, no.3, pp.305-324.

Swaminathan P. S., Lin P.-C.K., and Kathri S.P. (2012): Timing Aware Partitioning for Multi-FPGA Based Logic Simulation Using Top-Down Selective Hierarchy Flattening. *Proc. 30th IEEE International Conference on Computer Design*, pp. 153-158.

C. Kim, H. Shin, and Y. Yu (1995): Performance-driven circuit partitioning for prototyping by using multiple FPGA chips. *Proc. Design Automation Conference*.

K. Roy-Neogi and C. Sechen (1995): Multiple FPGA partitioning with performance optimization. *Proc. Field-Programmable Gate Arrays*, pp. 146 – 152.

Krishna C Nunna, F. Mehdipour and K. Murakami (2013): Early Stage Power Management for 3D FPGAs Considering Hierarchical Routing Resources. *Proc. 23rd ACM/IEEE Great Lakes Symposium on Very large Scale Integration*, pp. 31-36.

Kostas Siozios and Dimitrios Soudris (2010): A Temperature-Aware Placement and Routing Algorithm Targeting 3D FPGAs. *Proc. IFIP AICT*, pp. 211-231.

V. Betz and J. Rose (1997): VPR: A New Packing, Placement and Routing Tool for FPGA Research. *International Workshop Field Programmable Logic and Applications*, pp. 213–222.

Cristinel Ababei, Hushrav Mogal and Kia Bazargan (2006): Three-Dimensional Place and Route for FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Vol. 25, Issue 6, pp. 1132-1140.

A. Papanikolaou et al. (2011): *Three Dimensional System Integration: IC Stacking Process and Design*, Springer.

J. W. Joyner, P. Zarkesh-Ha, J. A. Davis, and J. D. Meindl (2000): A Three-Dimensional Stochastic Wire-Length Distribution for Variable Separation of Strata. *Proc. IEEE Int. Interconnect Technology Conference*, pp. 126–128.

D. H. Kim, S. Mukhopadhyay, and S. K. Lim (2009): TSV-aware Interconnect Length and Power Prediction for 3D Stacked ICs. *Proc. IEEE Int. Interconnect Technology Conference*, pp. 26–28.

T. Thorolfsson, K. Gonsalves, and P. D. Franzon (2009): Design Automation for a 3DIC FFT Processor for Synthetic Aperture Radar: A Case Study. *Proc. ACM Design Automation Conf.*, pp. 51–56.

D. H. Kim and and S. K. Lim (2010): Through-Silicon-Via-aware Delay and Power Prediction Model for Buffered Interconnects in 3D ICs. *Proc. IEEE Int. workshop on System Level Interconnect Prediction Conference*, pp. 25–32.

Trifunovic. A and Knottenbelt. J (2004): A Parallel Algorithm for Multilevel K-way Hypergraph Partitioning. *Proc. 3rd International Symposium on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, pp. 114 – 121.

G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar (1997): "Multi-level Hypergraph Partitioning: Applications in VLSI design. *Proc. ACM/IEEE Design Automation Conf. (DAC)*, pp. 526–529.

FreePDK 2013: http://www.eda.ncsu.edu/wiki/FreePDK.

Efficient Parallel Algorithms for the Maximum Subarray Problem *

Tadao Takaoka
Department of Computer Science
University of Canterbury
Christchurch, New Zealand

Abstract

Parallel algorithm design is generally hard. Parallel program verification is even harder. We take an example from the maximum subarray problem and and show those two problems of design and verification.

The best known communication steps for a mesh architecture for the maximum subarray problem is $2n-1$. We give a formal proof for the parallel algorithm on the mesh architecture based on Hoare logic. The main part of the proof is to establish several space/time invariants with three indices (i,j,k). The indices (i,j) pair specifies the invariant at the (i,j) grid point of the mesh and k specifies the k-th step in the computation. Then ignoring additive constants, the communication steps are improved to $(3/2)n$ steps and finally n steps, which is optimal in terms of communication steps. Also the first algorithm is implemented on a Blue Gene parallel computer and performance measurements conducted are shown.

1 Introduction

The maximum subarray problem is to find a rectangular subarray in the given (n,n)-two dimensional array that maximizes the sum in it. If the array elements are non-negative, we have the trivial solution of the whole array. Thus we subtract the mean value, or another anchor value depending on applications. This problem has wide applications from graphics to data mining. In graphics, the maximum subarray corresponds to a brightest spot in the given graphic image. In data mining, suppose we spread the sale amounts of some product on a two dimensional array classified by ages and annual income. Then the maximum subarray corresponds to the most promising customer range.

The typical algorithm by Bentley [2] takes $O(n^3)$ time on a sequential computer. This has been improved to slightly sub-cubic by Tamaki and Tokuyama [8], and Takaoka [7]. When the data is large, such as (1024, 1024) in graphics applications, those time complexities are prohibitive. This is more so, when we need to process video images in dynamic changing situations. An obvious choice is parallel computing. In Takaoka [7], a parallel implementation on a PRAM is discussed. As parallel computers such as GPU are becoming readily available, we need to devise a parallel algorithm implementable on those parallel computers. In Bae and Takaoka [4], Bae [3] a parallel algorithm was implemented on an (n,n)-two dimensional architecture based on the row-wise prefix sum. In this paper, we implement an algorithm based on the column-wise prefix sum with different data flow on a two-dimensional mesh architecture. See Fig. 1. Our algorithm performs the computation in $2n-1$ steps, where steps mean communication steps. Each cell executes a few statements per communication step. Thus our algorithm is cost optimal with respect to the prefix sum-based sequential algorithm. A parallel algorithm for the same problem was implemented on the BPS/CGM architecture, which has more local memory and communication capabilities with remote processors [1].

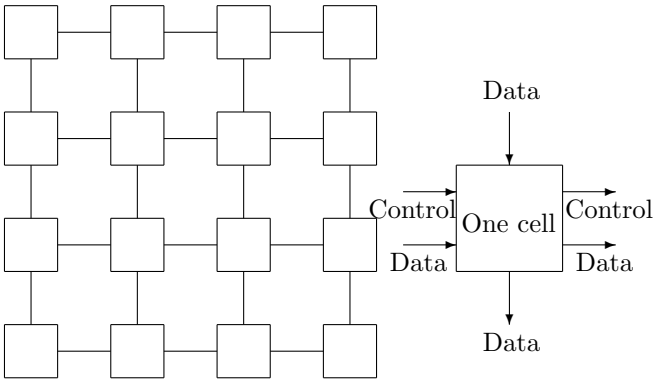


Figure 1: Two-dimensional architecture

We give a formal proof for the algorithm. It is based on the space-time invariants defined on the architecture. The proof leads us to a further speed-up of the computation. The data flow in the first implementation is from left to right and from top to down. The proof reveals that the processors to the right are idling at the the beginning. We first extend the data flow to operate in both directions horizontally to get the $(3/2)n$ steps result. Then we further extend data flow to operate in both directions vertically, i.e., data flow in four directions, so that the solution can be obtained at the center. By this method we achieve n steps, which is optimal. Algorithms are given by pseudo code.

*This research was supported by the EU/NZ Joint Project, Optimization and its Applications in Learning and Industry (OptALI).

2 Sequential algorithm

We modify a sequential algorithm based on row-wise prefix sums introduced in [4] to the one based on column-wise prefix sums, to develop Algorithms 1, 2 and 3.

Algorithm - sequential

```
for i := 1 to n do
{min_sum[i][0] := ∞; sum[i][0] := 0;
  sol[i][0] := −∞; }
S := −∞;
fork := 1 to n do {
  for j := 1 to n do column[k − 1][j] := 0;
  for i := k to n do {
    for j := 1 to n do{
      column[i][j] := column[i − 1][j] + a[i][j];
      sum[i][j] := sum[i][j − 1] + column[i][j];
      min_sum[i][j] :=
        min{sum[i][j], min_sum[i][j − 1]; };
      max_sum[i][j] := sum[i][j] − min_sum[i][j];
      sol[i][j] := max{max_sum[i][j], sol[i][j − 1]};
    } /* j */
      if solution[i][n] > S then S := solution[i][n];
  } /* i */
} /* k */
```

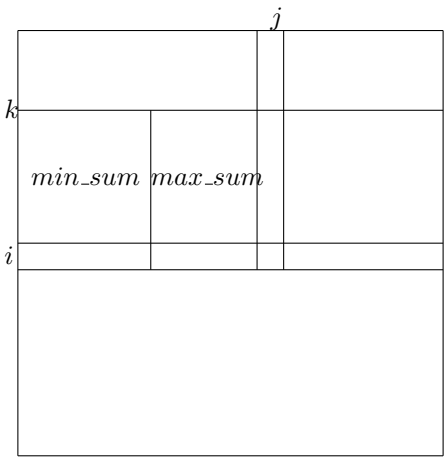


Figure 2: Strip-based sequential computation

The computation proceeds with the strip of the array from position k to position i. The variable $column[i][j]$ is the sum of array elements in the j-th column from position k to position i in array a. The variable $sum[i][j]$, called a prefix-sum, is the sum of the strip from position 1 to position j. Within this strip, variable j sweeps to compute $column[i][j]$ by adding $a[i][j]$ to $column[i-1][j]$. Then the prefix sum of this strip from position 1 to position j is computed by adding $column[i][j]$ to $sum[i][j-1]$. The variable $min\_sum[i][j]$ is the minimum prefix sum of this strip from position 1 to position j, expressed by "min_sum" in the figure. If the current sum is smaller than $min\_sum[i][j]$, $min\_sum[i][j]$ is replaced by it. $sol[i][j]$ is the maximum sum in this strip so far found from position 1 to position j. It is computed by taking the maximum of $sol[i][j-1]$ and $sum[i][j] - min\_sum[i][j]$, expressed by "max_sum" in the figure. After the computation for this strip is over, the global solution, S, is updated by $sol[i][n]$. This computation is done for all possible i and k, taking $O(n^3)$ time.

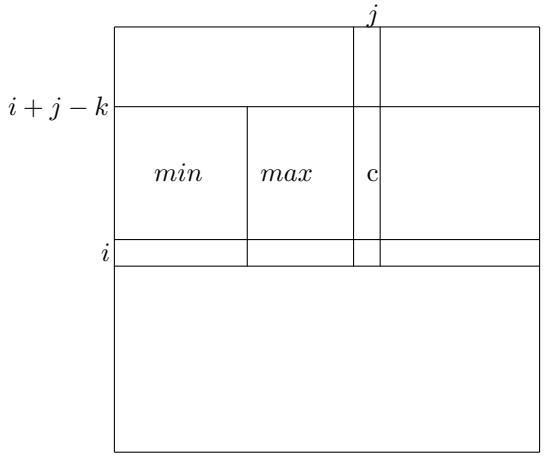


Figure 3: Illustration for Algorithm 1 (c for $column$)

3 Parallel algorithm Algorithm 1

The following is a parallel algorithm corresponding to the sequential algorithm in the previous section. The following program is executed by a cell at the (i, j) grid point. Each $cell(i, j)$ is aware of its position (i, j). Data flow is from left to right and from top to down. The control signals are fired at the left border, and propagate right. When the signal arrives at the cell (i, j), it accumulates the column sum "$column$" (c in the figure), the sum "sum", and update min_sum, etc. We assume all corresponding instructions in all cells are executed at the same time, that is, they are synchronized. We will later make some comments on asynchronous computation.

Algorithm 1

```
Initialization
for all i, j between 0 and n do in parallel
  {column[i][j] := 0; min[i][j] := ∞;
  control[i][[j] := 0; sum[i][j] := 0;}
for all i in parallel do {control[i][0] = 1;
  sol[i][0] := −∞; }
Main
  for k := 1 to 2n − 1 do
    for all i, j between 1 and n do in parallel {
      if control[i][j − 1] = 1 then {
        column[i][j] := column[i − 1][j] + a[i][j];
        sum[i][j] := sum[i][j − 1] + column[i][j];
        min[i][j] :=
          minimum(min[i][j − 1], sum[i][j]);
        max[i][j] := sum[i][j] − min[i][j];
        sol[i][j] := maximum(sol[i − 1][j],
          sol[i][j − 1], sol[i][j], max[i][j]);
        control[i][j] := 1;
      }
    } ** i, j **
  } ** k **
```

We prove the correctness of this parallel program in a framework of Hoare logic [5] based on a restricted form of that in Owicki and Gries [6]. The latter is too general to cover our problem. We keep the minimum extension of Hoare logic to our mesh architecture. The meaning of $\{P\}S\{Q\}$ is that if P is true before program (segment) S and S stops, then Q is true after S stops. The typical loop invariant appears as that for a while-loop; "while B do S". Here S is a program, and B is a Boolean condition. If we can prove $\{P \wedge B\}S\{P\}$, we can conclude $\{P\}$ while B do $S\{P \wedge \sim B\}$, where $\sim B$ is the negation of B. P is called the loop invariant, because P holds whenever

the computer comes back to this point to evaluate the condition B. This is time-wise invariant as the computer comes back to this point time-wise. We establish invariants in each cell. They are regarded as time-space invariants because the same conditions hold for all cells as computation proceeds. Those invariants have space indices i and j, and time index k. Thus our logical framework is a specialization of Owicki and Gries to indexed assertions.

The main assertions are given in the following. Note the difference between the wordings "sum of" and "sum in". Indices are attached to assertion names when necessary.

At time k (at the end of the k-th iteration) the following hold.
For $i = 1, ..., n$ and $j = 1, ..., k$
P_0 : $control[i][j] = 1$
P_1 : $c[i][j]$ is the column sum of $a[i+j-k, ..., i][j]$, that is, the sum of the j-th column of array a from position $i + j - k$ to position i
P_2 : $s[i][j]$ is the sum of $a[i + j - k, ..., i][1, ..., j]$
P_3 : $min[i][j]$ is the minimum of $s[i][l], l = 1, ..., j$
P_4 : $max[i][j]$ is the maximum of the sum of $a[i + j - k, ..., i][l, ..., j], 1 \le l \le j$.
P_5 : $sol[i][j]$ is the maximum sum in $a[i + j - k, ..., i][1, ..., j]$, that is, the sum of the maximum subarray of array portion $a[i + j - k, ..., i][1, ..., j]$.

The above are equivalent to
$1 \le i \le n \wedge 1 \le j \le k \Rightarrow P_0, ..., P_5$.
From this we obviously have $P_0(k) = true, ..., P_5(k) = true$ for $k = 0$.

For each P_0, ..., P_5 we omit indices i and j. Using the time index k, we prove $\{P_0(k-1)\}cell(i,j)\{P_0(k)\}$, ..., $\{P_5(k-1)\}cell(i,j)\{P_5(k)\}$.

We use the following proof rules. Let x_1, ..., x_n be local variables in $cell(1), ..., cell(n)$. There can be several in each cell. We use one for simplicity. The meaning of y_i/x_i is that the occurrence of variable x_i in Q is replaced by y_i. Parallel execution of $cell(1), ..., cell(n)$ is shown by $[cell(1)||...||cell(n)]$.

Parallel assignment rule

$$\frac{P \Rightarrow Q[y_1/x_1, ..., y_n/x_n], \quad \{Q[y_1/x_1, ..., y_n/x_n]\}cell(i)\{Q\}\textbf{for} i = 1, ..., n}{\{P\}[cell(1)||...||cell(n)]\{Q\}}$$

Other programming constructs such as composition (semi-colon), if-then statement, etc. in sequential Hoare logic can be extended to the parallel versions. Those definitions are omitted, but the following rule for for-loop for the sequential control structure, which controls a parallel program S from outside, is needed for our verification purpose.

Rule for for-loop

$$\frac{\{P(0)\}, \{P(k-1)\}S\{P(k)\}}{\{P(0)\}\textbf{for} k := 1 \textbf{to } n \textbf{ do } S\{P(n)\}}$$

This P represents P_0, ..., P_5 in our program. S is a parallel program $[cell(1)||...||cell(n)]$. Each $cell(i)$ has a few local variables and assignment statements. For an arbitrary array x, we regard $x[i][j]$ as a local variable for $cell(i, j)$. A variable from the neighbour, $x[i-1][j]$, for example, is imported from the upper neighbour. Updated variables are fetched in the next cycle. How to implement this part depends on the parallel computing environment used. See Section 6. The proof for each $\{P(k-1)\}cell(i,j)\{P(k)\}$ for $P = P_0, ..., P_5$ is given in Appendix.

THEOREM 1 *Algorithm 1 is correct. The result is obtained at $cell(n, n)$ in $2n - 1$ steps.*

Proof. From the second rule for a sequential loop, we have $P_5(2n-1)$ at the end.
$P_5(2n-1)$ at $cell(n, n)$
$\Leftrightarrow$ $sol[n][n]$ is the maximum sum in $a[n + n - 2n + 1, ..., n][1, ..., n]$
$\Leftrightarrow$ $sol[n][n]$ is the maximum sum in $a[1, ..., n][1, ..., n]$.

4 Algorithm 2

This algorithm does communication bi-directionally in a horizontal way. For simplicity we assume n is even. The (n, n) mesh is divided into two halves, left and right. The left half operates in the same way as Algorithm 1. The right half operates in a mirror image, that is, control signals go from right to left initiated at the right border. All other data also flows from right to left. At the center, that is, at $(i, n/2)$, $cell(i, n/2)$ performs "$center[i] := max[i][n/2] + max[i][n/2 + 1]$", which adds the two values that are the sums of strip regions in the left and right whose heights are equal and thus can be added to form a possible solution over the center. At the end of the k-th iteration, all assertions in Algorithm 1 hold on the left half and the assertions in mirror image hold on the right half. In addition, we have that $center[i]$ is the value of the maximum subarray that lies above or touching the i-th row and crosses over the center line.

Algorithm 2

```
Initialization
for all i, j between 0 and n do in parallel
  {column[i][j] := 0; min[i][j] := −∞;
    control[i][[j] := 0; sum[i][j] := 0;}
for all i do in parallel
  {control[i][0] := 1; control[i, n + 1] := 1; }
Main
  for k := 1 to (3/2)n − 1 do
    for all i = 1, ..., n, j = 1, ..., n do in parallel
      if 1 ≤ j ≤ n/2 then /*** left half ***/
        if control[i][j − 1] = 1 then {
          column[i][j] := column[i − 1][j] + a[i][j];
          sum[i][j] := sum[i][j − 1] + column[i][j];
          min[i][j] :=
              minimum(min[i][j − 1], sum[i][j]);
          max[i][j] := sum[i][j] − min[i][j];
          sol[i][j] := maximum(sol[i − 1][j],
            sol[i][j − 1]); sol[i][j], max[i][j]);
          control[i][j] := 1;
        }
      if n/2 + 1 ≤ j ≤ n then /*** right half ***/
        if control[i][j + 1] = 1 then {
          column[i][j] := column[i − 1][j] + a[i][j];
          sum[i][j] := sum[i][j + 1] + column[i][j];
          min[i][j] :=
              minimum(min[i][j + 1], sum[i][j]);
          max[i][j] := sum[i][j] − min[i][j];
          sol[i][j] :=
            maximum(sol[i − 1][j], sol[i][j + 1]),
            sol[i][j], max[i][j]);
          control[i][j] := 1;
        }
      if j = n/2 then {
      /*** cell(i, n/2) processes center[i] ***/
        center[i] := max[i][n/2] + max[i][n/2 + 1];
        if center[i] < center[i − 1] then
          center[i] := center[i − 1];
      }
```

```
    } / ** i, j **/
  } /** k **/
***Finalization step ***
  Let cell(n, n/2) do
    solution = maximum(sol[n][n/2],
                       sol[n][n/2 + 1], center[n]);
```

The strip $cell(i, j)$ processes is $a[i + j - k, ..., i][1, ..., j]$ in the left half and that in the right half is $a[i + n - j + 1 - k, ..., i][j, ..., n]$. Thus the cell $cell(i, n/2)$ and $cell(i, n/2 + 1)$ process the strips of the same height in the left half and the right half. Communication steps are measured by the distance from $cell(1, 1)$ to $cell(n, n/2)$, or equivalently from $cell(1, n)$ to $cell(n, n/2 + 1)$, which is $(3/2)n - 1$. By adding the finalization step, we have $(3/2)n$ for the total communication steps.

5 Algorithm 3

In this algorithm data flows in four directions. The array is divided into two halves; left and right as in the previous section. Column sums c and prefix sums s accumulate downwards as before, whereas column sums d and prefix sums t accumulate upwards. See Figure 4.

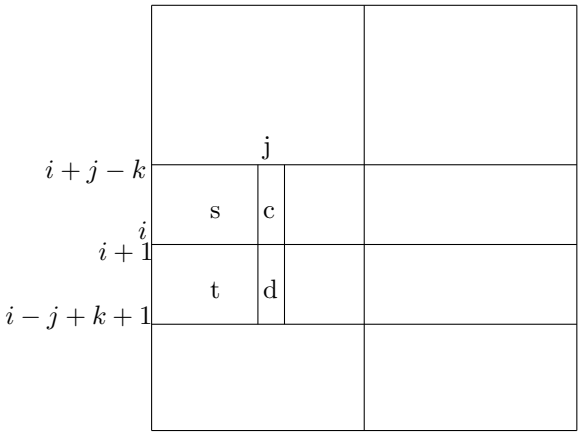


Figure 4: Illustration for Algorithm 3

The proof of Algorithm 1 reveals that at the end of the k-th iteration, $s[i][j]$ is the sum of $a[i + j - k, ..., i][1, ..., j]$ and $t[i+1][j]$ is the sum of $a[i+1, ..., i-j+k+1][1, ..., j]$. The height of each subarray is $k - j+1$. Since the width of those two areas are the same, we can have the prefix sum $u[i][j] = s[i][j]+t[i+1][j]$ that covers $a[i + j - k, ..., i - j + k + 1][1, ..., j]$, the height of which is $2(k - j + 1)$. That is, spending k steps, we can achieve twice as much height.

The solution array sol is calculated as before, but the result is sent into three directions; up, down and right in the left half and up, down and left in the right half. We have the invariant that $sol[i][j]$ is the maximum sum in subarray $a[i+j-k, ..., i-j+k+1][1, ..., j]$ in the left half. Substituting $i = n/2, j = n/2$, and $k = n - 1$ yields the subarray $a[1, ..., n][1, ..., n/2]$. Similarly $sol[n/2][n/2 + 1]$ is the maximum sum in the subarray $a[1, ..., n][n/2 + 1, ..., n]$. For simplicity we deal with the maximum subarray whose height is an even number. For a general case, see the note at the end of this section.

Algorithm 3

```
Initialization
for all i, j between 0 and n + 1 do in parallel {
  c[i][j] := 0; d[i][j] := 0;
  min[i][j] := −∞; control[i][[j] := 0;
  s[i][j] := 0; t[i][j] := 0;}
for all i in parallel do
  {control[i][0] := 1; control[i, n + 1] := 1; }
Main
for k := 1 to n − 2 do
  for all i = 1, ..., n, j = 1, ..., n do in parallel
    if 1 ≤ j ≤ n/2 {
      if control[i][j − 1] = 1 then {
        c[i][j] := c[i − 1][j] + a[i][j];
        s[i][j] := s[i][j − 1] + c[i][j];
        d[i][j] := d[i + 1][j] + a[i][j];
        t[i][j] := t[i][j − 1] + d[i][j];
        u[i][j] = s[i][j] + t[i + 1][j];
        min[i][j] := minimum(min[i][j−1], u[i][j]);
        max[i][j] := u[i][j] − min[i][j];
        sol[i][j] := maximum(sol[i − 1][j],
          sol[i + 1][j]), sol[i][j − 1], sol[i][j]);
        sol[i][j] := maximum(sol[i][j], max[i][j]);
        control[i][j] := 1;
      }
      if n/2 + 1 ≤ j ≤ n {
        if control[i][j + 1] = 1 then {
          c[i][j] := c[i − 1][j] + a[i][j];
          s[i][j] := s[i][j + 1] + c[i][j];
          d[i][j] := d[i + 1][j] + a[i][j];
          t[i][j] := t[i][j + 1] + d[i][j];
          u[i][j] := s[i][j] + t[i + 1][j];
          min[i][j] := minimum(min[i][j+1], u[i][j]);
          max[i][j] := u[i][j] − min[i][j];
          sol[i][j] := maximum(sol[i − 1][j],
            sol[i + 1][j], sol[i][j + 1]), sol[i][j]);
          sol[i][j] := maximum(sol[i][j], max[i][j]);
          control[i][j] := 1;
        }
      }
      if j = n/2 {
      /*** cell(i, n/2) performs the following. ***/
        center[i] := max[i][n/2] + max[i][n/2 + 1];
        if center[i] < center[i − 1] then
            center[i] = center[i − 1];
        if center[i] < center[i + 1] then
            center[i] := center[i + 1];
      }
    } /** i, j **/
} ** k **
if i = n/2 and j = n/2 then
/** cell(n/2, n/2) processes solution **/
  solution := maximum(sol[n/2][n/2],
                sol[n/2][n/2 + 1], center[n/2];
```

The computation proceeds with $n - 2$ steps by k and the last step of comparing the results from $cell(n/2, n/2)$ and $cell(n/2+1, n/2)$, resulting in $n-1$ steps in total.

Note. We described the algorithm for the solution whose height is an even number. This fact comes from the assignment statement "$u[i][j] := s[i][j] + t[i + 1][j]$" where the height of subarrays whose sums are s and t are equal. To accommodate a height of an odd number, we can use the value of t one step before, whose height is one shorter. To accommodate such odd heights, we need to almost double the size of the program by increasing the number of variables.

6 Implementation issues

We implemented Algorithm 1 on the Blue Gene/P computer under the MPI/Parallel C program environment. There are many practical issues to be considered. We summarize just three issues here as representatives.

We can let each $cell(i,j)$ know its position (i,j) by a system call "MPI_Cart_coords".

The next issue is synchronization. We assumed the corresponding statements in all cells are executed in a synchronized way. If we remove this assumption, that is, if the execution goes in asynchronous manner, the algorithm loses its correctness. Most mesh computers run asynchronously, but have synchronization primitives. Suppose we have the simplest synchronization primitive "synchronize". This means that when all cells come to this primitive, they can go ahead. In MPI, this primitive is called "MPI_Barrier". To make a correct program, we double the number of variables, that is, we prepare variable x_1 for every variable x. Let us associate the space/time index, (i,j,k) with each variable. Let us call $x(i,j,k)$ the current variable and the variable with indices different by one a neighbour variable. For example $sol[i][j]$ in the right-hand side of the assignment statement is a time-wise neighbour and that at the left-hand side is a current variable. Also $column[i-1][j]$ in the rght-hand side is a neighbour variable space-wise and time-wise, and so on. If x is a current variable, change it to x_1. If it is a variable of a neighbour keep it as it is. Let us call the modified program P_1. Now we define "update" to be the set of assignment statements of the form $x := x_1$.

Example Let P be a one-dimensional mesh program given below, which shifts array x by one place. Let us suppose $x[0] = 0$ and $x[i]$ are already given.

P : for all i do in parallel $x[i] := x[i-1]$;

Here $x[i]$ is the current variable and $x[i-1]$ is a neighbour variable space-wise and time-wise. An asynchronous computer can make all values 0. For the intended outcome, we perform P_1, synchronize and *update*.

P_1 : for all i do in parallel $x_1[i] := x[i-1]$;
synchronize;
update : for all i do in parallel $x[i] := x_1[i]$;

For our mesh algorithm, Algorithm 1, omitting the initialization part, we make the program of the form.

for $k := 1$ **to** $2n-1$ **do**
beginP_1; *synchronize*; *update* **end**.

The third implementation issue is related to the number of available processors. As the number of processors is limited, for large n we need to have what is called a coarse grain parallel computer. This means that given a large (n,n)-array, each processor is given its territory. Suppose, for example, we are given $(1024, 1024)$ array and only 16 processors are available. The input array is divided into sixteen $(256, 256)$ subarrays, to which the sixteen processors are assigned. Let us call the subarray for each processor its territory. Each processor simulates one step of Algorithm 1 sequentially. These simulation processes by sixteen processors are done in parallel. At the end of each simulation, the values in the registers on the right and bottom border are sent to the left and top borders of the right neighbour and the lower neighbour. The simulation of one step takes $O((n/p)^2)$ time, and $2n-1$ steps are carried out, meaning the computing time is $O(n^3/p^2)$ at the cost of $O(p^2)$ processors. When $p = 1$, we hit the sequential complexity of $O(n^3)$. If $p = n$, we have the time complexity of Algorithm 1, which is $O(n)$.

Based on the above methods for implementation, we implemented Algorithm 1 on the Blue Gene computer at the University of Canterbury. The version was BlueGene/P with 4096 processors, called cores. For the software side, the programming environment of MPI and the parallel C compiler, mpixlc, were used with optimization level 5. The timing results are shown below. The unit of time is second. (n,n) random arrays are tested. The time for generating uniformly distributed random numbers is not included in the time measurement. The mesh architecture can be figured into a 2-D mesh or 3-D mesh. We figured it into 2-D and included the configuration time in the measurement. The data items were loaded appropriately into processors and loading time was excluded from time measurement. As we can see from the table below, for small n, the configuration time dominates and a large number of processors has no effect. As the size of array increases, however, the speed increases with a large number of processors.

n	$p^2 = 1$	$p^2 = 16$	$p^2 = 25$	$p^2 = 100$
50	0.03275	0.00627	0.01274	0.03378
100	0.05143	0.02742	0.02869	0.04643
200	0.14794	0.12909	0.10898	0.11304
500	1.21697	1.32867	1.09698	2.29324
1000	7.72829	8.62130	6.28106	2.96971
1500	22.3838	27.8474	19.1569	7.24681
2000	49.7604	63.9644	42.7302	15.1321
2500	95.3663	120.392	82.2593	26.7386

7 Lower bound

Algorithms 2 and 3 are not very efficient for practical purposes. Rather their roles are to show the optimal bound of communication steps. Suppose we have a value a in the top-left cell and b in the bottom-right cell. All others are -1. Obviously the solution is a if $a > b$, and b otherwise. The values a and b need to meet somewhere. It is easy to see the earliest possibility is at time n-1. Thus Algorithm 3 is optimal in communication steps. The role of Algorithm 2 is a bridging step to Algorithm 3.

8 Concluding remarks

We gave a formal proof to a parallel algorithm for the maximum subarray problem. The formal proof for the other two parallel algorithms can be given in a similar way, but the details are omitted. The formal proof is not only for the reliability of the algorithm, but also it clarifies what is going inside the algorithm. In fact, the other two parallel algorithms, Algorithm 2 and Algorithm 3, have been developed by the insight into the data flow, given by the formal proof of Algorithm 1. We simplified the proof by synchronizing everything. The asynchronous version with (synchronize, update) in Section 6 would require about twice as much complexity for verification since we double the number of variables. Once the correctness of the synchronized version is established, that of the asynchronous version will be acceptable without further verification.

The algorithms are of fine-grain in the sense that each array element, or pixel in graphics, is processed by a processor. When the given array is large, such as (1024, 1024), this is not practical. That is, we need to develop a parallel algorithm of coarse grain. This means one processor will take care of some portion of the given array. When each processor finishes one step of k, the time index, it can communicate with

the neighbour for data transmission. In fact this version has been implemented on the BlueGene parallel computer with 4096 processors, and we observed a remarkable speed-up with 100 processors. The experiment conducted was rather of small scale. A larger experiment with a larger number of processors will be carried out in the future.

If we analyze dynamic images, such as video images, data loading becomes a big issue. Data must come through the top or left border processors. For the sake of speed, data must be processed in a pipeline manner, that is, data must be fed into the mesh architecture while the previous image is still being processed. We already have this pipe-lined version of Algorithm 1, which must be tested on the BlueGene for time measurement.

In our architecture, communication with the right neighbour and left neighbour cannot be done at the same time, that is, they are done one by one. To speed up Algorithm 2 and Algorithm 3, we need a more advanced architecture with bi-directional communication capabilities.

Appendix

Proof for $\{P_0(k-1)\}cell(i,j)\{P_0(k)\}$. At the beginning of the k-th iteration, $control[i][j] = 1$ for $j = 1, ..., k-1$, equivalently $control[i][j-1] = 1$ for $j = 1, ..., k$. $cell(i,j)$ performs "$control[i][j] := 1$" for $j = 1, ..., k$. Thus we have $\{P_0(k-1)\}cell(i,j)\{P_0(k)\}$ for $j = 1, ..., k$.

Proof for $\{P_1(k-1)\}cell(i,j)\{P_1(k)\}$. At time $k-1$, $c[i-1][j]$ is the column sum of $a[i-1+j-(k-1), ..., i-1][j] = a[i+j-k, ..., i-1][j]$. "$c[i][j] := c[i-1][j] + a[i][j]$" is performed for $i = 1, ..., n$ and $j = 1, ..., k$ in parallel. Thus $\{P_1(k-1)\}cell(i,j)\{P_1(k)\}$ holds.

Proof for $\{P_2(k-1)\}cell(i,j)\{P_2(k)\}$. At time $k-1$, $s[i][j-1]$ is the sum of $a[i+j-1-(k-1), ..., i][1, ..., j-1] = a[i+j-k, ..., i][1, ..., j-1]$. At time k, "$s[i][j] := s[i][j-1] + c[i][j]$" is performed for $i = 1, ..., n$ and $j = 1, ..., k$ in parallel. Thus $s[i][j]$ is the sum of $a[i+j-k, ..., i][1, ..., j]$, and $\{P_1(k-1)\}cell(i,j)\{P_1(k)\}$ holds.

Proof for $\{P_3(k-1)\}cell(i,j)\{P_3(k)\}$.
At time $k-1$, $min[i][j-1]$ is the minimum of $s[i][l], l = 1, ..., j-1$. At time k, "$min[i][j] := minimum(min[i][j-1], s[i][j])$" is performed for $i = 1, ..., n$ and $j = 1, ..., k$ in parallel. Thus $min[i][j]$ is the minimum of $s[i][l], l = 1, ..., j$. Therefore $\{P_3(k-1)\}cell(i,j)\{P_3(k)\}$ holds.

Proof for $\{P_4(k-1)\}cell(i,j)\{P_4(k)\}$. At time k, $min[i][j]$ is the minimum of $s[i][l], l = 1, ..., j$. At time k, "$max[i][j] := s[i][j] - min[i][j]$" is performed for $i = 1, ..., n$ and $j = 1, ..., k$ in parallel. Thus $max[i][j]$ is the maximum of the sum of $a[i+j-k, ..., i][l, ..., j]$ for $1 \le l \le j$. Therefore $\{P_4(k-1)\}cell(i,j)\{P_4(k)\}$ holds.

Proof for $\{P_5(k-1)\}cell(i,j)\{P_5(k)\}$.
At time $k-1$, $sol[i][j-1]$ is the maximum sum in
$a[i+j-1-(k-1), ..., i][1, ..., j-1]$
$= a[i+j-k, ..., i][1, ..., j-1]$,
and $sol[i-1][j]$ is the maximum sum in
$a[i-1+j-(k-1), ..., i-1][1, ..., j]$
$= a[i+j-k, ..., i-1][1, ..., j]$.
At time k,
$sol[i][j] := maximum(sol[i-1][j], sol[i][j-1], sol[i][j], max[i][j])$
is performed for $i = 1, ..., n$ and $j = 1, ..., k$ in parallel. The first two cases do not cover $a[i][j]$. The last two cases cover $a[i][j]$. $sol[i][j]$ is the solution for $cell(i,j)$ for the time up to $k-1$, which does not cover row $i+j-k$, and $max[i][j]$ is the maximum sum of the strip that ends at column j. Thus $sol[i][j]$ is the maximum sum in $a[i+j-k, ..., i][1, ..., j]$, and $\{P_5(k-1)\}cell(i,j)\{P_5(k)\}$ holds.

Acknowledgment The author is thankful to Robin Candy who implemented Algorithm 1 on the Blue Gene and conducted time measurements. He also appreciates useful discussions on the maximum subarray problem with Sung Eun Bae. Finally he is grateful to the supercomputer centre of University of Canterbury, Blue Fern, who offered a free use of Blue Gene for the research on the mesh algorithms.

References

[1] C. E. R. Alves, E. N. Caceres and S. W. Song: BPS/CGM Algorithms for Maximum Subsequence and Maximum Subarray, EuroPVM/MPI 2004, LNCS 3241: 139-146 (2004)

[2] Jon Louis Bentley: Perspective on Performance. Commun. ACM 27(11): 1087-1092 (1984)

[3] Sung Eun Bae: Sequential and Parallel Algorithms for Generalized Maximum Subarray Problem. Ph. D thesis. University of Canterbury (2007)

[4] Sung Eun Bae and Tadao Takaoka: Algorithms for the Problem of K Maximum Sums and a VLSI Algorithm for the K Maximum Subarrays Problem. I-SPAN 2004: 247-253 (2004)

[5] C.A.R. Hoare. An axiomatic basis for computer programming. Communications of the ACM, 12(10):576-580 (1969).

[6] S. Owicki and D. Gries: Verifying properties of parallel programs: An axiomatic approach. Communications of the ACM, 19(5):279-285 (1976)

[7] Tadao Takaoka: Efficient Algorithms for the Maximum Subarray Problem by Distance Matrix Multiplication. Electr. Notes Theor. Comput. Sci. 61: 191-200 (2002)

[8] Hisao Tamaki, Takeshi Tokuyama: Algorithms for the Maximum Subarray Problem Based on Matrix Multiplication. SODA 1998: 446-452 (1998)

Communication Delegation Method for Exascale Systems

Yugendra R. Guvvala[1] **Yu Zhuang**[2]

Department of Computer Science
Texas Tech University,
Lubbock, Texas-79409, USA
[1]Email: yugendra.r.guvvala@ttu.edu

[2] Email: yu.zhuang@ttu.edu

Abstract

High Performance Computing is trending towards exascale and some of the major barriers of high performance computing or scientific computing are dominated by latencies incurred due to storage, communication, and component failures. In this paper we discuss a technique to overcome one of those obstacles: latency incurred due to communication. This programming technique is developed using existing MPI and OpenMP communication models and the technique discussed in this paper is Communication Delegation Method, which will provide an efficient way of communicating across nodes and reducing communication channel resource contention.

Keywords: Communication Delegation, Multicore Machines, Communication Channel, Resource Contention, Exascale Computing

1 Introduction

In today's Scientific and high-performance computing era Researchers, Scientists, and Engineers require High Performance Computing resources to solve many complex problems. Thus, deployments of huge high performance computing resources are growing at a rapid rate. These deployments are growing in size and competing on speed by increasing number of nodes and cores. According to Moor's Law projection and projections from the trend of the Top500 listings we can observe that next generation computing is cruising towards the exascale systems we will approach it not later than end of this decade.

There are various methodologies and programming models which are evolving to identify parallelism in scientific applications. This evolution is leading towards highly scalable and embarrassingly parallel applications. The scalability of these applications leads towards a very complex communication patterns, thus requiring high bandwidth for better performance. Many scientific computing applications in critical areas of research, such as chemistry, nanotechnology, astrophysics, climate, and high-energy physics are becoming more and more communication intensive. Some of them such as gene sequencing or other bioinformatics applications are not very sensitive to latencies, where as some applications such as weather, oil and gas need low latency, high bandwidth to perform better. There are several approaches made to improve communication bandwidth and reduce latencies. In next generation machines, which might have tens to hundreds of cores per node even a small communication overhead per core would add up to a huge latency. In this study we look at the application and identify its communication needs and delegate communication to communication cores to reduce latency.

2 Background

There has been several studies on communication methods, programming models and hardware enhancements (2)(8)(9) which contribute to improvement of communication mechanism on High Performance Computing clusters. Some of these contributions are software based where as others are hardware based. We will look into some of the existing methodologies here.

2.1 Software Based Approaches

Software based approaches exist from early days of cluster computing, some of these approaches has been evolving based on the hardware enhancements and faster interconnects. But, primitive and standard approach for exchanging messages across processors is MPI (1). MPI is a standard for communication across multiple processors and this felicitates development of parallel applications and libraries. Other implementations of communication models by targeting a specific application stack and hardware are implementation of MPICH(4), MVAPICH(6), OpenMP(3) and Global Arrays(5).

From above we can see that there are several implementations which are software based but depend on many specifications either accommodating specific hardware or application. The current and future generation of HPC machines are complex mixture of different hardware based on budget and requirements. Implementation of both shared and distributed memory. Thus neither MPI (it's implementations) nor Open MP, are a de facto standard communication methodology.

2.2 Hardware Based Approaches

In early methods to improve performance of the communication on clusters some of them approach by incorporating message transfer into memory controllers of the system(9) some try to integrate communication into processor internals(10)(11), some design and implement lean communication software layers, where as other techniques isolate

communication processes to a dedicated communication core .

Recent improvements are mostly based on complex and faster interconnects, like Infiniband(7), Gigabit Ethernet(12), Tofu Interconnect(13), and other interconnects in high performance computing(14), Still this interconnects are not standard as they depend on vendor for their implementation purpose. Thus developing applications for specific underlying hardware is not the best idea.

The direction of Exascale system is towards multicores and many nodes model. Due to multiple architectures, we need to identify a consistent communication design approach which would fit in and scale applications appropriately over different deployments. This signifies the need of dynamic method which can adopt to different types of hardware and application requirements.

3 Communication Delegation Method

3.1 Motivation

Due to different communication processes & characteristics, and hardware developments, we need to develop programming models which are efficient and flexible to adopt towards next generation exasclae machines. As throughput of many scientific applications is dependent on communication process we need to identify techniques to adopt it in next generation systems.

Considering multicore architectures we identify that communication between processes on cores belonging to same die is faster than communication of processes on cores from remote processors. The communication channel is a resource for multiple cores this channel gets over whelmed and raises communication channel resource contention issue.

3.2 Design Rationale

Communication delegation models is a idea of grouping cores based on there spatial locality, dedicating a core per group for communication purpose. For application with high communication intensity and high scalability we need special considerations. Next generation machines will be equipped with tens to hundreds of cores per node, thus the we need to adopt an appropriate communication methodology (MPI, OpenMP, or Hybrid) based on application and system needs. All existing models are efficient but, non of the models address communication channel contention issue. In the methods discussed in this paper we try to reduce the impact of this problem. Compute cores (red) perform computations where as Communication core (green) gathers data to be transferred in a pool and packages it and transfers the data to communication core on another communication pool.

i) Design Example: From the Fig 1 we can see that a set of cores form a group , called communication pool and we have a delegated core for communication purpose.

ii) MPI Communication Delegation Method: This model explains how we can implement the above discussed design in a MPI based application. In this design we make use of communication worlds of MPI, we group communication pools

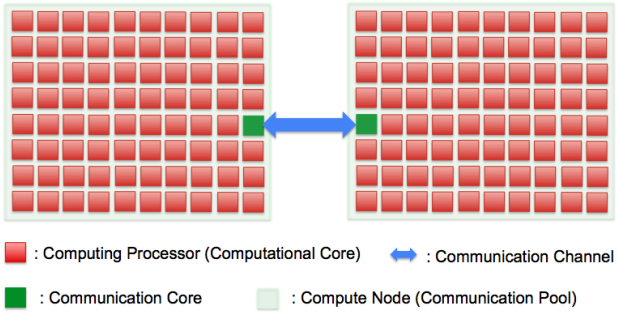


Figure 1: Design Example: 80 core node with one delegated core for communication

into different communication worlds, we group all the delegated cores for communication into a another set of communication world. This allows delegated cores to pass messages across pools.

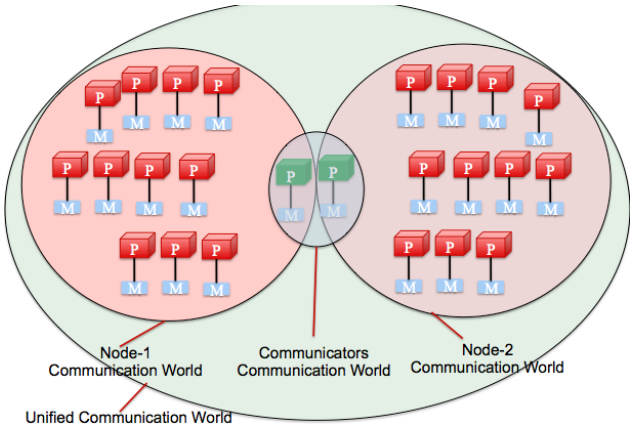


Figure 2: MPI Communication Delegation Method

In Fig 2 we used the communication worlds of MPI to form different groups for communication. This communication world pools would allow us to gather communications of all the processes within the pool and transferring them at a time to another pool's communication core.

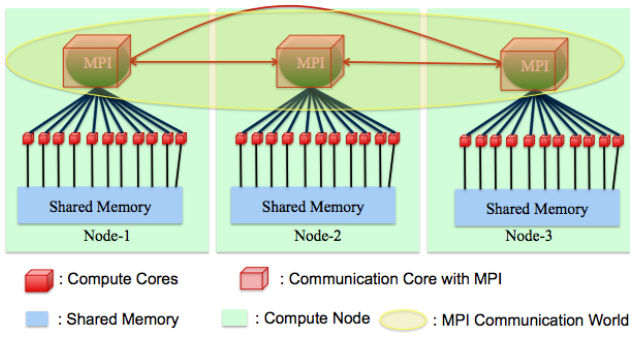


Figure 3: Hybrid Communication Delegation Model

iii) Hybrid Communication Delegation Method: This design is targeting the Hybrid programming model using both MPI and OpenMP. In a hybrid model we have inter node communication using OpenMP and intra node communication using MPI. But if we have a core dedicated for communication purpose which also acts as a master node for the communication pool to spawn slave threads, and perform other administrative tasks such as gathering and

buffering communications and work as a fail-over core for the pool. Form Fig 3 we can observe that inter node cores form communication pools and delegated cores form a pool use MPI communication world to communicate with each other.

4 Theoretical Validation

We validate the proposed methodology using both hardware and software evaluation.

4.1 Hardware Validation:

Considering the fastest FSB and fastest PCIe busses we can see that the FSB is at least 2 times faster than the PCIe, thus we can see that if we send an information to a closer core by placing it on to a memory location it is faster than transferring across the node using the PCIe card and a interconnect. In Fig 4 you can look at different internal buses. Every core has a faster access bus to the memory, where as there is only one communication channel per node to communicate intra nodes.

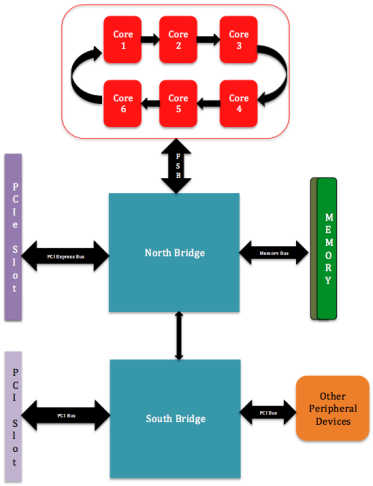


Figure 4: Different internal Communication Channels (Bus) of a System

There are some existing hardware which already provide similar functionality. For example Mellonox Infiniband adopters offload the communication on tho the Infiniband PCIe adaptors. How ever in this model as we are delegating a core and inter core communication is faster compared to PCI bus communication this model proves to be more efficient.

4.2 Software Validation:

Different Non-data overheads (**?**) of MPI that can be reduced using CDM are as follows.
i) Basic MPI stack Overhead:
ii) Request Allocation or Queuing Overhead:
iii) Tag and Source Matching Overhead:
iv) Algorithmic Complexity and Multi-Request Operation:
v) Derived Datatype Processing:
vi) Buffer Alignment Overhead:
vii) Unexpected Message Overhead:
viii) Thread Communication Overhead:

5 Experimental Evaluation

In this section we demonstrate the performance and right choice of applications based on communication characteristics. We will look at two major classical examples which are building block of many scientific and high-performance applications, Matrix Multiplication, and Fast Fourier Transforms (FFT).

For performance evaluation of this method we use a DELL cluster powers by PowerEdge R410 Blade servers. The cluster consists of 96 nodes with 2x sockets and intel Hex-core (Intel Xeon X5650) 2.66 GHz processors. Compute nodes were running Linux CentOS 5.4 cluster operating system. Each node contained a Mellanox Infiniband Host Channel Adapter (HCA) supporting 4x Quad Data Rate (QDR) connections with a speed of 20 Gbps. Each node also has 48 GB DDR3 1333 Mhz of memory the Cluster is a 12.3 TeraFLOPS.

5.1 Matrix Multiplication:

Fig-5 shows performance of different sizes of matrix multiplication. Comparison is made between MPI Implementation, Hybrid (OpenMP + MPI) implementation, Communication Delegation Method (CDM) incorporated MPI implementation and Communication Delegation Method hybrid implementation. Three different matrix sizes results are reported in this paper. The test bed of implementation consisted of four compute nodes with 12 cores each in a communication pool with 1 delegated processor. All tests were run on a 4 node or 48 cores with 4 communication pools (one per each node) and 4 communication cores (one per each pool/node).

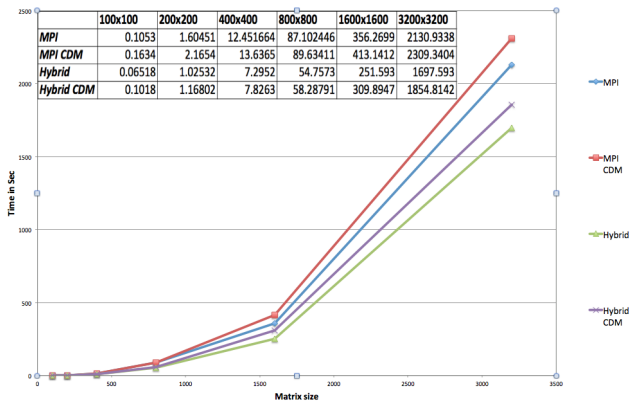


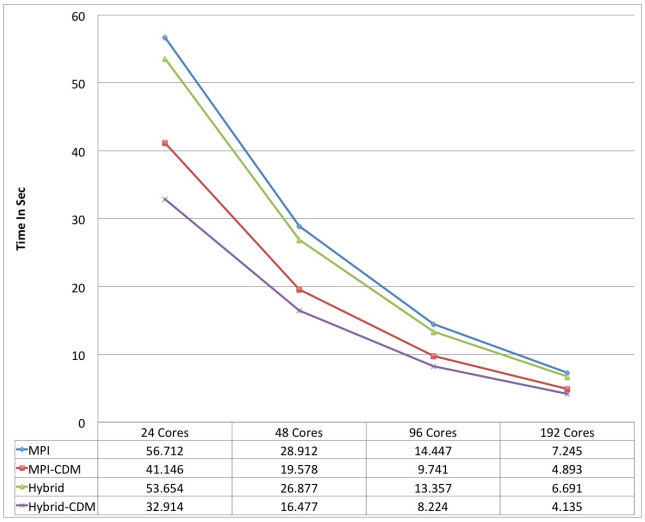
Figure 5: Matrix Multiplication with increasing Size of Matrix

From Fig-5, product of A and B multiplicand matrices of sizes 100x100 , 200x200, 400x400, 800x800, 1600x1600 and 3200x3200 elements each are computed. We can see form the graph that Matrix multiplication which has less communication is not the best candidate for adopting this model.

5.2 NAS Parallel Benchmark-FT:

The NAS parallel benchmarks (NPB) were developed by NASA Ames research centre for performance evaluation of parallel and distributed computers. After considering different characteristics of the various benchmark kernels we identified FT (Fourier Transforms) kernel to be most appropriate kernel for evaluating CDM.

The Fast Fourier Transform (FT) benchmark solves a partial differential equation (PDE) using forward and inverse FFTs. The most interesting part for us was the 3D FFTs (NxNxN grid size) which play a key role in this benchmark and requires considerable amount of communication for operation such as array transposition. The steps involved in this kernel are initially it initialized the input array using a psudorandom generator, followed by a forwards 3-D FFT computation, then the transformed data is multiplied in loop with a coefficient array followed by inverse 3-D FFT computation. FT kernel is programmed in both MPI and OpenMP version in Fortran with around 20 MPI routines. We rewrote this benchmark tool with both Open MP and MPI based Communication Delegation Method (CDM) and observed good results.



	24 Cores	48 Cores	96 Cores	192 Cores
MPI	56.712	28.912	14.447	7.245
MPI-CDM	41.146	19.578	9.741	4.893
Hybrid	53.654	26.877	13.357	6.691
Hybrid-CDM	32.914	16.477	8.224	4.135

Figure 6: NAS NPB-3.1.1, FT Benchmark

The Fig-6 shows the performance of different implementations with time on different number of cores. We optimized bigger problem size kernel C(512^3) We can see that for a 24 core node it was huge performance difference as the application was communication heavy, eventually the performance deteriorates because number of cores dedicated for communication purpose increase. So if we have a high core density on a node like 100 cores per node and we dedicate only 1 core per node for communication purpose this would give us huge improvements. Thus this kind of applications on a Exascale machine would be estimated to perform outstandingly.

6 Conclusion and Future Work

So we conclude that we the model proposed in this paper has a huge impact on high core density machines which will have a contention issue on the communication channel resource. The model is more of a dynamic solution as we can change number of cores delegated for communication purpose based on application needs. The applications with his communication intensity are the best application for adopting CDM.

References

[1] Gropp, William, Ewing Lusk, Nathan Doss, and Anthony Skjellum. "A high-performance, portable implementation of the MPI message passing interface standard." Parallel computing 22, no. 6 (1996): 789-828.

[2] Martin, Richard P., Amin M. Vahdat, David E. Culler, and Thomas E. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. Vol. 25, no. 2. ACM, 1997.

[3] Dagum, Leonardo, and Ramesh Menon. "OpenMP: an industry standard API for shared-memory programming." Computational Science and Engineering, IEEE 5, no. 1 (1998): 46-55.

[4] Gropp, William. "MPICH2: A new start for MPI implementations." In Recent Advances in Parallel Virtual Machine and Message Passing Interface, pp. 7-7. Springer Berlin Heidelberg, 2002.

[5] Nieplocha, Jaroslaw, Robert J. Harrison, and Richard J. Littlefield. "Global Arrays: A portable shared-memory programming model for distributed memory computers." In Proceedings of the 1994 ACM/IEEE conference on Supercomputing, pp. 340-349. IEEE Computer Society Press, 1994

[6] Panda, D. K. "MVAPICH."

[7] InfiniBand Trade Association. InfiniBand Architecture Specification: Release 1.0. InfiniBand Trade Association, 2000.

[8] Buenabad-Chvez, Jorge, Miguel A. Castro-Garca, and Graciela Romn-Alonso. "Simple, list-based parallel programming with transparent load balancing." In Parallel Processing and Applied Mathematics, pp. 920-927. Springer Berlin Heidelberg, 2006.

[9] E. D. Brooks III, B. C. Gorda, K. H.Warren, and T.S. Welcome. BBN TC2000 Architecture and Programming Models

[10] J.B. Carter, A. Davis, R. Kuramkote, C. Kuo, L.B. Stoller, and M. Swanson. Avalanche: A Communication and Memory Architecture for Scalable Parallel Computing. Technical Report UUCS-95-022, University of Utah, 1995

[11] D. Chiou, B.S. Ang, Arvind, M.J. Beckerle, G.A. Boughton, R. Greiner, J.E. Hicks, and J.C. Hoe. StarT-NG: Delivering Seamless Parallel Computing. In EURO-PAR95 Conference, Aug. 1995.

[12] Cunningham, David, Bill Lane, and William Lane. Gigabit Ethernet Networking. Macmillan Publishing Co., Inc., 1999.

[13] Ajima, Yuuichirou, Tomohiro Inoue, Shinya Hiramoto, and Toshiyuki Shimizu. "Tofu: Interconnect for the K computer." Fujitsu Sci. Tech. J 48, no. 3 (2012): 280-285.

[14] Taubenblatt, Marc A. "Optical interconnects for high-performance computing." Lightwave Technology, Journal of 30, no. 4 (2012): 448-457.

[15] Pavan Balaji, Anthony Chan, William Gropp, Rajeev Thakur, and Ewing Lusk. 2010. The Importance of Non-Data-Communication Overheads in MPI. Int. J. High Perform. Comput. Appl. 24, 1 (February 2010), 5-15. DOI=10.1177/1094342009359528 http://dx.doi.org/10.1177/1094342009359528

Author Index

Recent Volumes in the CRPIT Series

ISSN 1445-1336

Listed below are some of the latest volumes published in the ACS Series *Conferences in Research and Practice in Information Technology*. The full text of most papers (in either PDF or Postscript format) is available at the series website `http://crpit.com`.

Volume 124 - Database Technologies 2012
Edited by Rui Zhang, The University of Melbourne, Australia and Yanchun Zhang, Victoria University, Australia. January 2012. 978-1-920682-95-8.
Contains the proceedings of the Twenty-Third Australasian Database Conference (ADC 2012), Melbourne, Australia, 30 January – 3 February 2012.

Volume 125 - Information Security 2012
Edited by Josef Pieprzyk, Macquarie University, Australia and Clark Thomborson, The University of Auckland, New Zealand. January 2012. 978-1-921770-06-7.
Contains the proceedings of the Tenth Australasian Information Security Conference (AISC 2012), Melbourne, Australia, 30 January – 3 February 2012.

Volume 126 - User Interfaces 2012
Edited by Haifeng Shen, Flinders University, Australia and Ross T. Smith, University of South Australia, Australia. January 2012. 978-1-921770-07-4.
Contains the proceedings of the Thirteenth Australasian User Interface Conference (AUIC2012), Melbourne, Australia, 30 January – 3 February 2012.

Volume 127 - Parallel and Distributed Computing 2012
Edited by Jinjun Chen, University of Technology, Sydney, Australia and Rajiv Ranjan, CSIRO ICT Centre, Australia. January 2012. 978-1-921770-08-1.
Contains the proceedings of the Tenth Australasian Symposium on Parallel and Distributed Computing (AusPDC 2012), Melbourne, Australia, 30 January – 3 February 2012.

Volume 128 - Theory of Computing 2012
Edited by Julián Mestre, University of Sydney, Australia. January 2012. 978-1-921770-09-8.
Contains the proceedings of the Eighteenth Computing: The Australasian Theory Symposium (CATS 2012), Melbourne, Australia, 30 January – 3 February 2012.

Volume 129 - Health Informatics and Knowledge Management 2012
Edited by Kerryn Butler-Henderson, Curtin University, Australia and Kathleen Gray, University of Melbourne, Australia. January 2012. 978-1-921770-10-4.
Contains the proceedings of the Fifth Australasian Workshop on Health Informatics and Knowledge Management (HIKM 2012), Melbourne, Australia, 30 January – 3 February 2012.

Volume 130 - Conceptual Modelling 2012
Edited by Aditya Ghose, University of Wollongong, Australia and Flavio Ferrarotti, Victoria University of Wellington, New Zealand. January 2012. 978-1-921770-11-1.
Contains the proceedings of the Eighth Asia-Pacific Conference on Conceptual Modelling (APCCM 2012), Melbourne, Australia, 31 January – 3 February 2012.

Volume 133 - Australian System Safety Conference 2011
Edited by Tony Cant, Defence Science and Technology Organisation, Australia. April 2012. 978-1-921770-13-5.
Contains the proceedings of the Australian System Safety Conference (ASSC 2011), Melbourne, Australia, 25th – 27th May 2011.

Volume 134 - Data Mining and Analytics 2012
Edited by Yanchang Zhao, Department of Immigration and Citizenship, Australia, Jiuyong Li, University of South Australia, Paul J. Kennedy, University of Technology, Sydney, Australia and Peter Christen, Australian National University, Australia. December 2012. 978-1-921770-14-2.
Contains the proceedings of the Tenth Australasian Data Mining Conference (AusDM'12), Sydney, Australia, 5–7 December 2012.

Volume 135 - Computer Science 2013
Edited by Bruce Thomas, University of South Australia, Australia. January 2013. 978-1-921770-20-3.
Contains the proceedings of the Thirty-Sixth Australasian Computer Science Conference (ACSC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 136 - Computing Education 2013
Edited by Angela Carbone, Monash University, Australia and Jacqueline Whalley, AUT University, New Zealand. January 2013. 978-1-921770-21-0.
Contains the proceedings of the Fifteenth Australasian Computing Education Conference (ACE 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 137 - Database Technologies 2013
Edited by Hua Wang, University of Southern Queensland, Australia and Rui Zhang, University of Melbourne, Australia. January 2013. 978-1-921770-22-7.
Contains the proceedings of the Twenty-Fourth Australasian Database Conference (ADC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 138 - Information Security 2013
Edited by Clark Thomborson, University of Auckland, New Zealand and Udaya Parampalli, University of Melbourne, Australia. January 2013. 978-1-921770-23-4.
Contains the proceedings of the Eleventh Australasian Information Security Conference (AISC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 139 - User Interfaces 2013
Edited by Ross T. Smith, University of South Australia, Australia and Burkhard C. Wünsche, University of Auckland, New Zealand. January 2013. 978-1-921770-24-1.
Contains the proceedings of the Fourteenth Australasian User Interface Conference (AUIC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 140 - Parallel and Distributed Computing 2013
Edited by Bahman Javadi, University of Western Sydney, Australia and Saurabh Kumar Garg, IBM Research, Australia. January 2013. 978-1-921770-25-8.
Contains the proceedings of the Eleventh Australasian Symposium on Parallel and Distributed Computing (AusPDC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 141 - Theory of Computing 2013
Edited by Anthony Wirth, University of Melbourne, Australia. January 2013. 978-1-921770-26-5.
Contains the proceedings of the Nineteenth Computing: The Australasian Theory Symposium (CATS 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 142 - Health Informatics and Knowledge Management 2013
Edited by Kathleen Gray, University of Melbourne, Australia and Andy Koronios, University of South Australia, Australia. January 2013. 978-1-921770-27-2.
Contains the proceedings of the Sixth Australasian Workshop on Health Informatics and Knowledge Management (HIKM 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 143 - Conceptual Modelling 2013
Edited by Flavio Ferrarotti, Victoria University of Wellington, New Zealand and Georg Grossmann, University of South Australia, Australia. January 2013. 978-1-921770-28-9.
Contains the proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling (APCCM 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 144 - The Web 2013
Edited by Helen Ashman, University of South Australia, Australia, Quan Z. Sheng, University of Adelaide, Australia and Andrew Trotman, University of Otago, New Zealand. January 2013. 978-1-921770-15-9.
Contains the proceedings of the First Australasian Web Conference (AWC 2013), Adelaide, Australia, 29 January – 1 February 2013.

Volume 145 - Australian System Safety Conference 2012
Edited by Tony Cant, Defence Science and Technology Organisation, Australia. April 2013. 978-1-921770-13-5.
Contains the proceedings of the Australian System Safety Conference (ASSC 2012), Brisbane, Australia, 23rd – 25th May 2012.