THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

A Slowdown Prediction Method to Improve Memory Aware Scheduling

Andreas de Blanche

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

A Slowdown Prediction Method to Improve Memory Aware Scheduling

Andreas de Blanche
ISBN: 978-91-7597-369-2

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr: 4050
ISSN: 0346-718X

Technical Report 123D
Department of Computer Science and Engineering
Research group: Division of Computer Engineering

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31 – 772 1000

Andreas de Blanche
Email: andreas@deblanche.se

Cover:

The cover image shows the bimodal distribution of all possible job-schedules from the experiment in **Paper IV**, Section 6, broken down into three normal distributions. The lines show the best, average, and worst slowdowns as well as the schedule predicted as the best by the presented slowdown prediction method.

Chalmers Reproservice
Göteborg, Sweden 2016

A Slowdown Prediction Method to Improve Memory Aware Scheduling

Andreas de Blanche
Division of Computer Science and Engineering
Chalmers University of Technology

ABSTRACT

Scientific and technological advances in the area of integrated circuits have allowed the performance of microprocessors to grow exponentially since the late 1960's. However, the imbalance between processor performance and memory bus capacity has increased in recent years. The increasing on-chip-parallelism of multi-core processors has turned the memory subsystem into a key factor for achieving high performance. When two or more processes share the memory subsystem their execution times typically increase, even at relatively low levels of memory traffic. Current research shows that a throughput increase of up to 40% is possible if the job-scheduler can minimize the slowdown caused by memory contention in industrial multi-core systems, such as high performance clusters, datacenters or clouds. In order to optimize the throughput, the job-scheduler has to know how much slower the process will execute when co-scheduled with other processes on the same server. Consequently, unless the slowdown is known, or can be fairly well estimated, the scheduling becomes pure guesswork and the performance suffers.

The central question addressed in this thesis is how the slowdown caused by memory traffic interference between processes executing on the same server can be predicted and to what extent. This thesis presents and evaluates a new slowdown prediction method which estimates how much longer a program will execute when co-scheduled on the same multi-core server as another program. The method measures how external memory traffic affects a program by generating different levels of synthetic memory traffic while observing the change in execution time. Based on the observations it makes a first order prediction of how much slowdown the program will experience when exposed to external memory traffic.

Experimental results show that the method's predictions correlate well with the real measured slowdowns. Furthermore, it is shown that scheduling based on the new slowdown prediction method yields a higher throughput than three other techniques suggested for avoiding co-scheduling slowdowns caused by memory contention. Finally, a novel scheme is suggested to avoid some of the worst co-schedules, thus increasing the system throughput.

Keywords: multi-core processor, slowdown-aware scheduling, memory bandwidth, resource contention, last-level cache, co-scheduling, performance evaluation.

IT ALL DEPENDS ON THE APPLICATION!

LIST OF APPENDED PAPERS

This thesis is based on the work contained in the following publications. References to the papers are made using the Roman numerals associated with the papers.

i. **Andreas Boklund**[1], Nima Namaki, Stefan Mankefors-Christiernin, Johan Gustafsson and Mikael Lingbrand, "Dual Core Efficiency for Engineering Simulation Applications", *Proceedings of the International conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, July 2008.

ii. **Andreas de Blanche** and Stefan Mankefors-Christiernin, "Method for Experimental Measurement of an Application's Memory Bus Usage", *Proceedings of the International conference on Parallel and Distributed Processing Techniques and Applications,* Las Vegas, USA, July 2010.

iii. **Andreas de Blanche** and Thomas Lundqvist, "A Methodology for Estimating Co-Scheduling Slowdown due to Memory Bus Contention on Multicore Nodes", *Proceedings of the International Conference on Parallel and Distributed Computing and Network,* Innsbruck, Austria, February 2014.

iv. **Andreas de Blanche** and Thomas Lundqvist, "Addressing Characterization Methods for Memory Contention Aware Co-scheduling", *The Journal of Supercomputing*: Volume 71, Issue 4, Page 1451-1483, Springer Verlag, 2015.

Books and book chapters not included in the thesis[1]:

- **A. de Blanche**, *Operating systems – Theory and Practice* (in Swedish), ISBN: 978-91-44-01980-2, Studentlitteratur, Lund, 2008.
- **A. Boklund,** "Assessing Lectures, a case Study on the Student's Perception", *Shifting Perspectives in Engineering Education*, Michael Christie (editor), ISBN: 91-631-8476-1, Chalmers Strategic Effort on Learning and Teaching, Chalmers University of Technology, pp 96-103, Göteborg, Sweden, 2006.

Peer-reviewed publications not included in the thesis[1]:

- P. Lindström, **A. de Blanche**, "Integration and Optimization of a 64-core HPC for FEM- and/or CFD Welding Simulations", *Improving Simulation Prediction by Using Advanced Material Models*, NAFEMS NORDIC, Lund, Sweden, 5 – 6 November 2013.
- **A. de Blanche**, N. Namaki, S. Mankefors-Christiernin, "Multicore Clusters for CFD-simulations, Comparative Study of Three CFD-Softwares", *Parallel and Distributed Techniques and Applications,* Las Vegas, USA, July 2012.

[1] Between 1975 and 2008 I was known as Andreas Boklund.

- N. Namaki, **A. de Blanche**, “Black-box Characterization of Processor Workloads for Engineering Applications”, *IEEE International Symposium on Workload Characterization*, Atlanta, USA, December 2010.
- **A. de Blanche**, S. Mankefors-Christiernin, “Availability of Unused Computational Resources in an Ordinary Office Environment”, *International Journal of Circuits Systems and Computers*, 19/3, pp. 557-572, 2010.
- **A. de Blanche**, S. Mankefors-Christiernin, “Minimizing Total Cost and Maximizing Throughput - A Metric for Node versus Core Usage in Multi-Core Clusters”, *Proceedings of the International conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 2010.
- N. Namaki, **A. de Blanche**, S. Mankefors-Christiernin, “Exhausted Dominated Performance - Basic Proof of Concept“, *Proceedings of the International conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 2010.
- N. Namaki, **A. de Blanche**, S. Mankefors-Christiernin, "A Tool for Processor Dependency Characterization of HPC Applications", *Proceedings of HPC Asia 2009*, Kaohsiung, Taiwan, 2009.
- N. Namaki, **A. de Blanche**, S. Mankefors-Christiernin, "Exhaustion Dominated Performance - A First Attempt", *Proceedings of the 24th Annual ACM Symposium on Applied Computing*, Honolulu, Hawaii, USA, 2009.
- **A. Boklund**, N. Namaki, S. Mankefors-Christiernin, C. Jiresjö, "A Characterization Tool for the Impact of Network Deficiencies of HPC Applications", *Proceedings of the International conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 2008.
- **A. Boklund**, S. Christiernin, C. Johansson, H. Lindell, "A Comparative Study of Forward and Reverse Engineering in UML Tools", *Proceedings of the International Conference on Applied Computing*, pp: 535-540, Salamanca, Spain, 2007.
- **A. Boklund**, “Performance Prediction of Future Generation Computer Systems”, *Proceedings of the International Conference on Applied Computing*, pp: 42-48, San Sebastian, Spain, 2006.
- G. Calås, **A. Boklund**, S. Mankefors-Christiernin, “A First Draft of RATF – A Method Combining Robustness Analysis and Technology Forecasting”, *Proceedings of the 3rd IEEE International Conference on Information Technology; New Generations*, pp: 72-77, Las Vegas, USA, 2006.
- C. Jiresjö, **A. Boklund**, “Benefits of Alternative Network Topologies for COTS Linux Clusters”, *Proceedings of the International Conference on Applied Computing*, pp: 457-461, San Sebastian, Spain, 2006.
- R. Persson, S. Mankefors-Christiernin, **A. Boklund**, “Concept Model of an Object Detecting Mobile Robot in an Indoor Environment”, *Proceedings of the International Conference on Applied Computing*, pp: 585-589, San Sebastian, Spain, 2006.

- L. G. Christiernin, M. Gustavsson, A. Ohlsson, **A. Boklund**, "Prototyping a Multi-Layered Help - a User Involved Exploratory Design", *Proceedings of the International Conference on Applied Computing*, pp: 315-323, San Sebastian, Spain, 2006.
- G. Calås, S. Mankefors-Christiernin, **A. Boklund**, "Robustness Analysis and Technology Forecasting: Survey on a Missing Combination in Software Development", *Proceedings of the International conference on Software Engineering*, pp: 329-334, Innsbruck, Austria, 2006.
- **A. Boklund**, C. Jiresjö, S. Mankefors-Christiernin, N. Namaki, L. Gustavsson-Christiernin, M. Ebbmar, "Performance of Network Subsystems for a Technical Simulation on Linux Clusters", *Proceedings of the International Conference on Parallel and Distributed Computing and Systems*, pp: 503-509, Phoenix, Arizona, USA, 2005.
- **A. Boklund**, C. Selvefors, "A Low Budget Approach to Distributed Automated Black-Box Testing", *Proceedings of the International Conference on Software Engineering Research and Practice*, pp: 302-308, Las Vegas, Nevada, USA, 2005.
- **A. Boklund**, N. Namaki, C. Jiresjö, S. Mankefors-Christiernin, L. G. Christiernin, "Applicability of Parallel File Systems for Technical Simulations: A Case Study", *Proceedings of the International Conference on Applied Computing*, pp: 155-160, Algarve, Portugal, 2005.
- S. Mankefors-Christiernin, **A. Boklund**, "Applying Operational Profiles to Existing Random Testing Results: Meeting the Square Root Efficiency Challenge", *Proceedings of the International Conference on Applied Computing*, pp: 233-244, Algarve, Portugal, 2005.
- G. Calås, S. Mankefors-Christiernin, **A. Boklund**, "A Case Study Evaluation of 11 Hypothetical Software Evolution Laws", *Proceedings of the International Conference on Software Engineering*, pp: 26-31, Innsbruck, Austria, 2005.
- S. Mankefors-Christiernin, **A. Boklund**, "Random Testing Automatization – Efficiency Optimization Using the Integral Formulation", *Proceedings of the International Conference on Software Engineering*, pp: 150-155, Innsbruck, Austria, 2005.
- S. Mankefors-Christiernin, **A. Boklund**, "Efficiency Increase in Random Testing Using MPI-Based Parallelization", *Proceedings of the International Conference on Software Engineering*, pp: 168-173, Innsbruck, Austria, 2005.
- **A. Boklund**, S. Mankefors-Christiernin, C. Jiresjö, N. Namaki, "COTS-Cluster Evolution During the Last Decade and Extrapolation into the Next", *Proceedings of the International Conference on Parallel and Distributed Computing and Network*, pp: 614-619, Innsbruck, Austria, 2005.
- S. Mankefors-Christiernin, **A. Boklund**, "Multiple Profile Evaluation Using a Single Test Suite in Random Testing", *Proceedings of the 15th IEEE International Symposium on Software Reliability Engineering*, pp: 283-294, Saint-Malou, France, 2004.

- L. Ekberg, S. Mankefors-Christiernin, **A. Boklund**, "Software Code Quality with UML Design Models", *Proceedings of the 4th Conference on Software Engineering Research and Practice in Sweden*, pp: 63-72, Linköping, Sweden, 2004.
- **A. Boklund**, S. Mankefors-Christiernin, C. Jiresjö, "31-Nights around Midnight: Compute Node Behavior in a Real World Metamorphosic Cluster", *Proceedings of the 2004 the International Conference on Applied Computing*, pp: 36-43, Lisbon, Portugal, 2004.
- **A. Boklund**, C. Jiresjö, S. Mankefors, "The Story Behind Midnight, a Part Time High Performance Cluster", *Proceedings of the International conference on Parallel and Distributed Processing Techniques and Applications*, pp: 173-178, Las Vegas, USA, 2003.
- S. Mankefors, R. Torkar, **A. Boklund**, "New Quality Estimations in Random Testing", *Proceedings of the 14th IEEE International Symposium on Software Reliability Engineering*, pp: 468-478, Denver, USA, Nov. 2003.
- **A. Boklund**, F. Larsson, "The Kluster II Project", *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, 2001.

DECLARATION OF CONTRIBUTION

All papers included in this thesis have multiple authors which made smaller or larger contributions. However, the initiative behind the papers, the initial study design, the majority of the analysis as well as the majority of the written text and figures was in all papers performed by me. Many aspects of the papers, including study and experiment design, analysis and presentation were discussed with and influenced by my supervisors.

A more detailed, yet brief, summary of the author contributions:

i. Performed the initial study design which then all co-authors made some smaller contributions to. The experiments were carried out almost exclusively by J. Gustafsson, M. Lingbrand, and N. Namaki on site at Volvo Aero. The final analysis and the writing were done by me, S. Christiernin and N. Namaki; names are given in order of influence.

ii. Performed the study design, planned and performed the experiments as well as the analysis and wrote the majority of the text. The study design, analysis and presentation were discussed with and influenced by S. Christiernin.

iii. Performed the study design, planned and performed the experiments as well as the analysis and wrote the majority of the text. The study design, analysis and presentation were discussed with and influenced by T. Lundqvist.

iv. Performed the study design, planned and performed the experiments as well as the analysis and wrote the majority of the text. The study design, analysis and presentation were discussed with and partly influenced by T. Lundqvist.

PREFACE

Around the millennia I got my M.Sc. from the University of Gothenburg and started to work with engineering simulations at MSC.Software as a Linux and High Performance Computing (HPC) specialist. A few years later I joined the Computer Engineering division at University West, Sweden and I have been there ever since. During my tenure at the university I have continued to work with industrial companies and their engineering simulations on a project-to-project basis. I have not done any commercial projects for over a year now but when this dissertation is finished – who knows.

At University West I met Stefan Christiernin and in 2005 we obtained a grant for a project named "Enhanced Scalability of Simulations in Industrial Manufacturing" from the Swedish Knowledge Foundation and Nima Namaki joined us as a Ph.D. student. In that year I was also appointed assistant Ph.D. supervisor for Linn Gustavsson-Christiernin. She has successfully defended her thesis, ”Layered Design”, and I am very pleased to see that she has established herself as a research scientist at the Production Technology Center in Trollhättan.

In 2008 Nima successfully defended his Licentiate thesis and we secured funding for the follow up project ”Increased Simulation Efficiency in Industrial Manufacturing”. Nima is currently working in the field as a HPC architect. When Thomas Lundqvist joined the university we worked on a few projects together and in 2013 I decided that I really had to pick myself up, put all other projects aside, and finish my journey towards a Ph.D. in computer engineering.

And here we are!

To conclude I would like to thank my supervisors Stefan Christiernin, Thomas Lundqvist, and Per Stenström for their encouragement and guidance. I would also like to thank Nima Namaki for all the fun we had while working together and for discussing and dissecting all my wild ideas. Furthermore, I would like to thank Christian Jiresjö, Linn Gustavsson-Christiernin and Eric Lind, it was an honour to be in of the same research group as you guys and I wish you all the best in your future endeavours. Finally, I want to thank Göran Thorsell, Jonas Karlsson, Christian Lund, Niklas Andersson, and Susanne Ahlin at GKN Aerospace (former Volvo Aero and Volvo IT), it was a joy working with you and I hope I will have the pleasure of doing so again.

This work has been funded in part by the Swedish Knowledge Foundation (Stiftelsen för kunskaps och kompetensutveckling), grants 2005/0274 and 2008/0519 and by Volvo Aero Corporation, Volvo IT, Ansys Sweden and Medeso which all were involved in the enhanced scalability for industrial simulations projects together with University West.

Table of Contents

1. Introduction

In 1994 Donald Becker and Thomas Sterling created the now famous Beowulf cluster [2]. The initial system was constructed from 16 Intel 486 processor-based computers interconnected by channel bonded 10 megabit Ethernet and it was running Linux. The machine became an instant success and only ten years later around 60% of the top 500 supercomputers in the world were categorized as *clusters* [23]. Furthermore, in recent years, the number and size of public and private cloud data centers have increased quite dramatically. Cluster and cloud systems typically have hundreds, thousands or tens of thousands of interconnected servers concurrently executing an order of magnitude more processes. In these systems a so called job-scheduler performs the task of allocating processes to servers.

Up until the introduction of the multi-core-processors the far most common allocation scheme was to schedule one process to run on one server, which had one core. With the introduction of the first mainstream dual-core processors, the pure computational capacity of each cluster server practically doubled at little or no extra cost; and then came the multi-core processors. The cluster job-schedulers could now co-schedule several processes on the same server by allocating one process to each of the server's cores. Although process co-scheduling generally increases the overall throughput of a cluster or cloud system it also creates new challenges.

Processes executing on different cores in the same server typically share many of the server's resources such as, caches, buses and storage devices. When processes share a resource their execution is slowed down compared to if they would have had exclusive access to that resource. When two purely computationally bound processes are co-scheduled on the same server the slowdown will be close to zero. However, if for example two I/O bound processes are co-scheduled the slowdown can be over a factor of two [26]. If the slowdown is above two it is more efficient to execute the processes sequentially, both in terms of execution time and throughput. At the same time, if one purely computational process is co-scheduled with an I/O bound process the slowdown can be zero, since no resources are shared. Furthermore, studies [22, 26] have shown that the slowdown can be significant even at low levels of shared resource usage.

Consequently, the way in which processes are combined when allocated to servers can greatly influence the performance of both the processes and the overall system throughput. To minimize the process slowdowns, maximize the system throughput or both, the scheduler has to be able to estimate how much slower the program(s) will execute when co-scheduled with other program(s) and allocate processes to servers accordingly. Unless the slowdown is known, or can be fairly well estimated, the scheduling becomes pure guesswork and as a result the performance suffers.

Although there are many shared resources in a server, previous studies [9, 14, 20] have shown that the memory system is the scarcest resource for many programs. The reason for this is the increasing level of on-chip parallelism in combination with the limited off-chip bandwidth in modern multi-core processors; this problem is sometimes referred to as the *memory wall*

[25]. According to Tang et.al. [22], a throughput increase of 40% is possible in datacenters when scheduling programs based on memory bandwidth usage. Furthermore, in a recent study performed by Xu, et.al. [26], two co-scheduled programs experience a super-linear slowdown due to memory traffic interference. Hence, having the ability to predict the co-scheduling slowdown caused by memory bus sharing, and being able to co-schedule processes accordingly, is a key factor for achieving high performance in any cluster or cloud system.

In current research, different approaches for detecting and quantifying memory bus contention have been suggested, in order to avoid process slowdown and increase system throughput. These approaches are mainly based on last level cache (LLC) or memory bandwidth metrics. Several different cache based metrics have been suggested for memory contention aware scheduling. Daci and Tartari [6] compare the last level cache (LLC) miss rate metric with other characterization metrics and draw the conclusion that the LLC miss rate outperforms the other metrics. Zhuravlev et al. [27] as well as [4] have studied cache contention-aware scheduling techniques, such as stack distance competition [5], and their ability to mitigate the performance degradation of co-scheduled programs.

In addition to the cache based metrics several studies have reported on significant throughput increases when co-scheduling jobs based on memory bandwidth usage. Koukis and Koziris [12] implemented a memory bandwidth aware Linux kernel scheduler which co-schedules processes to keep the aggregated bandwidth usage below the maximum bus bandwidth. In [26], Xu et al. propose an operating system scheduling policy that first determines at which level of memory traffic the performance degradation is acceptable, and then co-schedules processes to keep the aggregated bandwidth usage below this level. Mars et al. [15] and Eklov et al. [8] have taken this approach one step further and apply a black-box profiling approach where they create contention in the memory system while measuring how it affects the program in terms of quality of service or execution time. The results can then be used as an input to a cluster or cloud job-scheduler.

Much of the research in this area has been focused on operating system schedulers and not on cluster or cloud job-schedulers. While an OS-scheduler performs in the order of thousand scheduling decisions each second a cluster or cloud job-scheduler allocates a process to a server where it might run for minutes, hours, weeks or months. Moving running processes between servers is an expensive operation and not always possible. Thus, readjusting the scheduling decisions continuously during execution is generally not an option in these environments.

Many techniques have been proposed to enhance the co-scheduling capabilities of memory contention aware cluster and cloud schedulers. However, despite significant advances, it is important to develop new techniques to increase the performance of memory contention aware job-schedulers for cluster and cloud systems.

1.1 Problem Statement

This thesis focuses on the slowdown that is caused by resource sharing between programs co-scheduled on the same server. Specifically, this thesis addresses the following problems:

- Although resource sharing between concurrently executing processes in a cluster-, grid- or cloud-system is a serious source of inefficiencies the potential of resource aware scheduling on industrial engineering simulations software have not been studied.
- Processes concurrently executing on different cores in the same processor typically share the memory bus which leads to execution slowdowns. An open question is how the slowdown caused by memory traffic interference between co-scheduled processes can be predicted and to what extent.
- While process co-scheduling generally increases the overall throughput of a cluster, grid or cloud system, it also creates new challenges. The important questions are how should processes be co-scheduled to minimize the overall slowdown and how much can the system throughput be improved by slowdown aware co-scheduling.

1.2 Research Contributions

This thesis makes several contributions within the area of resource aware job-scheduling with a focus on a new slowdown prediction method for efficient memory bus contention aware co-scheduling. The contributions are presented in four papers referred to as **Paper I**, **II**, **III** and **IV**. The contributions presented in the thesis are outlined below.

- The first contribution is a metric for exploring the impact resource sharing has on engineering simulation efficiency in an industrial multi-core cluster with regards to execution time as well as hardware and license costs. The metric is presented in **Paper I** where it is used to quantify the cost of different allocation strategies.
- The second contribution is a new method which estimates the slowdown programs experience when executing concurrently on different cores in the same processor. The method is first demonstrated in **Paper II**, it is then described in **Paper III** and enhanced in **Paper IV**. The method experimentally estimates a program’s memory bus usage by generating different levels of synthetic traffic while observing how the execution time is affected. Based on observations and interpolation it is possible to make a first order prediction of how much slowdown the program will experience when executing concurrently with a program that generates a given amount of memory traffic.
- The third contribution is the construction of a prototype tool that is used by the new slowdown based method. The first version of the tool is presented in **Paper II** and it has been enhanced with the additions made to the method in **Paper III** and **Paper IV**.
- The fourth contribution is the identification of trade-offs and relative effectiveness of four different methods aimed at enhancing the performance of memory resource aware scheduling. While these four methods have been studied in isolation, **Paper IV** is the first to compare their relative performance. The four methods are last-level cache miss rate, stack distance competition, memory bandwidth usage, and the slowdown based method. The most important findings of the paper are that the slowdown based method introduced in this thesis

performs better than the other methods and that using the last-level cache miss rate method actually decreases the overall throughput.

- The fifth contribution is a novel scheme that effectively avoids some of the worst co-scheduling combinations, thus increasing the throughput. **Paper IV** establishes a high occurrence of a certain type of co-scheduled pairs among the schedules with the worst throughput. It is observed that co-scheduling two instances of the same program, even if they have an extremely low or no slowdown, often degrades the overall system throughput.

1.3 Dissertation Organization

The contributions of the thesis are published in four papers referred to as **Paper I-IV**. A more in depth explanation of each individual contribution is presented successively in Sections 2 to 7.

Section 2 presents the first contribution (**Paper I**) which explores the impact resource sharing has on engineering simulation efficiency in an industrial cluster with regards to execution time as well as hardware and license costs. The second contribution, presented in Section 3, is a method that estimates the slowdown a program experiences when sharing the memory bus with other program(s) executing on the same server. The slowdown prediction method is exemplified in **Paper II**, described in **Paper III**, and enhanced in **Paper IV**. Section 4 contains the third contribution, and revisits **Paper II**, which is the design and implementation of the prototype tool used by the method for slowdown characterization of programs. Section 5 highlights contribution four from **Paper IV** where trade-offs and relative effectiveness of four different methods aimed at enhancing the performance of memory resource aware scheduling are compared. Section 6 presents the fifth contribution which is another aspect of **Paper IV** that proposes a novel scheme for cancelling the worst co-scheduling combinations. Finally, Section 7 provides concluding remarks and also considers future work.

In the following sections I have used the notations and terminology from ***Paper IV*** *which sometimes deviate from the earlier notations used in the appended publications.*

2 Dual-Core Efficiency for Engineering Simulations

Typically, a job-scheduler in a cluster or cloud system allocates one process to each core to exploit the additional computational capacity it provides. Nevertheless, the remaining server resources will be shared between the co-scheduled processes. As shown in [26], said resource sharing might limit the performance to such an extent that the computational benefits of the additional cores are eradicated. A common application of high performance computing in an industrial setting is technical engineering simulations. These are often parallel multi-process FEM[1] or CFD[2] simulations, which are computationally intensive but the computations are performed on large matrices which has to be transferred to and from memory. Furthermore, the processes perform intra- and inter-node communications to exchange data and synchronize the execution between each simulated time step.

This motivates investigating the most efficient allocation strategy for parallel CFD-simulations in an industrial dual-core cluster. That is, will the second core available in each server increase the efficiency of the CFD-simulations? In **Paper I** we use two metrics to determine the overall efficiency: *execution time* and a resource based *cost metric*.

Measuring and comparing the execution time of programs that execute for more than 1000 seconds on a Linux system is trivial. However, to evaluate the different allocation strategies based on cost we created the following metric:

$$Cost\ of\ job = \left(\frac{H}{N} * P + L * P\right) * \frac{R_{job}}{1\ year}$$

Eq. 1

The unit of the job cost associated with Eq. 1 is the currency used when deriving the input values of *H* and *L*. *H* represents the hardware cost for one of the servers during one year, this includes hardware acquisition as well as maintenance, localities, staff, energy, etc., and *N* is the number of processes executing on each server. *L* is the license cost of executing one process of the software for one year and *P* represents the degree of parallelization, i.e. the number of processes that will be used to execute the simulation. Finally, *R* is the run time of the job when divided into *P* processes. Basically, the left factor calculates the cost associated with executing the job for a full year. The left term is then multiplied with the right term which is the fraction of a year the current job will execute for.

[1] Finite Element Method (FEM) simulations are used by engineers to, among other things, simulate crack propagation, bending stresses and crash deformations.

[2] Computational Fluid Dynamics (CFD) simulations, are used by engineers to, among other things, simulate air flowing through a turbine or water flowing through a pipe.

The measurements and calculations in **Paper I** were carried out, on site, at an aerospace company using two commercial CFD programs, three different simulation models and a dual-core cluster with 220 servers. Furthermore, the hardware to license cost ratio for this setup was 1 to 15, i.e. the cost one single processor license was 15 times larger than the cost associated with maintaining one server for one year including a third of the acquisition costs, since the servers are exchanged every three years.

Paper I first observes that the executions are slowed down when co-scheduling *P* processes on *P/2* servers compared to when *P* processes are allocated to *P* servers. The paper then establishes that in all but one instance there is an execution time increase of between 2% and 38% when two processes are co-scheduled on the same server. However, the hardware requirements are halved when processes are co-scheduled. The most interesting finding in **Paper I** is that in five cases (out of 36) the CFD-simulation actually execute slower when two processes are co-scheduled on each server compared to when only one process are executing on each server, for the same number of physical servers. As an example, executing one of the five simulations in parallel using eight processes, scheduling one process on each of eight servers is faster than executing the same simulation using 16 processes and eight servers. The paper deduces that this is not caused by a faulty parallelization algorithm.

Paper I quantifies the cost of the different allocation strategies using the metric presented in Eq. 1. The paper then concludes that although the hardware usage is reduced when two processes are co-scheduled the jobs still cost more to execute due to the large disparity between hardware and license costs. In conclusion the scheme used in **Paper I** shows that in almost all cases it is more cost efficient to schedule one CFD process on each dual-core server than using co-scheduling due to slowdowns caused by resource sharing. In the case at hand the average cost efficiency is increased by 16.5%. The results also show that it is sometimes more cost efficient to increase the degree of parallelization.

Paper I proposes a scheme to estimate the efficiency of different allocation strategies in terms of execution time and cost. Furthermore, when the hardware to license ratio is known, the scheme presented in **Paper I** can be used to calculate the difference in execution time for which the allocation strategies have the same cost.

3 The Slowdown Prediction Method

Processes executing on different cores in the same computer typically share many of the server's resources such as caches, buses, and other resources. The resource sharing slows down the execution compared to when the processes have exclusive resource access. Although there are many shared resources in a server, previous studies [3, 22] have shown that memory bus sharing is a major source of process slowdown [1, 13, 16, 22]. Xu et al. [26] even observed super-linear slowdowns for processes sharing the memory bus.

Avoiding co-scheduling slowdowns caused by memory bus sharing is a key factor for attaining high performance in any cluster or cloud system. However, in order to avoid co-scheduling slowdowns they must first be known. To this end we propose a slowdown prediction method that is able to estimate how much slower any given program will execute when co-scheduled with any other program(s) due to memory bus contention. The method is exemplified in **Paper II**, described in **Paper III**, and enhanced in **Paper IV**. It is based on the hypothesis that it is possible to decouple a program's sensitivity to outside memory traffic (the slowdown it experience) from the *pressure* it places on other programs (the slowdown it causes in co-scheduled programs).

A brute-force approach to estimate the slowdown of co-scheduled processes is to execute all programs together with all other programs in all possible combinations and calculate the slowdown compared to that of a solo execution. However, this brute-force characterization method is not a feasible alternative since the number of combinations to evaluate scales as $O(N^C)$ where N is the number of programs and C is the number of cores sharing the resources.

Paper III addresses the brute-force method's high complexity by introducing a slowdown prediction method which execute each program a pre-determined number of times. The method is based on the hypothesis that if two programs, A and B, are co-scheduled, the memory traffic generated by program A will cause program B to execute slower and likewise the memory traffic generated by program B will cause program A to execute slower. To this end the slowdown prediction method first determines the amount of memory traffic (*pressure*) a program introduces into the system. The method then determines how sensitive a program is to competing memory traffic by generating synthetic traffic while executing the program and measuring how this affects the execution time. Hence, we introduce the concept of slowdown as a function of the memory bandwidth introduced into the system by co-scheduled processes, $S_{prog}(p)$ where p is the pressure, i.e. the amount of memory traffic introduced into the system by the memory traffic generator or programs co-scheduled with *prog*. The pressure is given as a fraction of the available memory bus bandwidth.

$$\textit{Slowdown of program prog} = S_{prog}(p) \qquad \text{Eq. 2.}$$

The slowdown of $S_{prog}(p)$ is given by subjecting the program *prog* to L different levels of memory traffic and using interpolation to estimate intermediate values. Thus, the complexity becomes $L \times N$ instead of N^C, where N is the number of programs and L is the number of memory traffic levels and C is the number of cores.

In **Paper III** and **Paper IV** the sensitivity function $S_{prog}(p)$ is based on linear interpolation based on five measurement points corresponding to 100% (max), 75%, 50%, 25%, and 0% of the sustainable memory bandwidth as reported by STREAM triad [17]. Figure 1 contains an example sensitivity curve where the measurement points are marked by X.

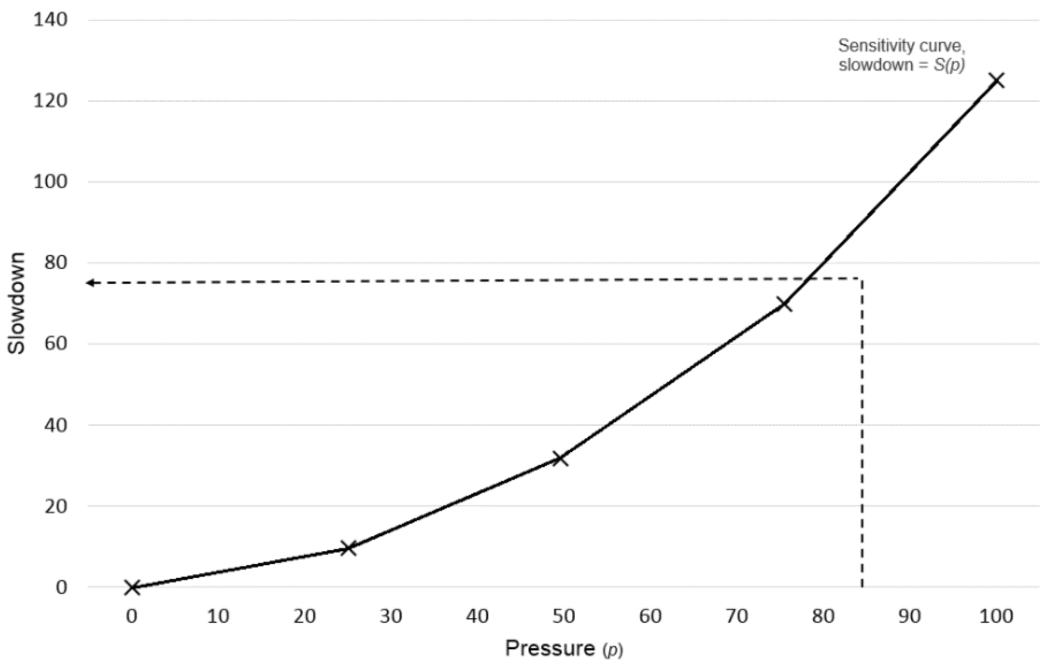

Figure 1: Slightly modified example sensitivity curve from Paper IV showing the slowdown for a program Y at different levels of pressure. If another program, *X*, introduces a pressure of *p_X= 85* the sensitivity function of program Y will evaluate to *$S_Y(P_X)$* equals 78.

If two programs A and B are co-scheduled on the same server the average job slowdown of that server would be expressed as Eq. 3 with *n=2*, where p_A and p_B represents the *pressure* that programs *A* and *B* introduces into the system and *n* is the number of co-scheduled processes.

$$\frac{S_A(p_B) + S_B(p_A)}{n}$$

Eq. 3.

Furthermore, if program *A* were to be co-scheduled with two instances of program *B*, on a computer with three or more cores, we would express this as Eq. 4, where *n=3*:

$$\frac{S_A(2p_B) + 2S_B(p_A + p_B)}{n}$$

Eq. 4.

In **Paper II** the pressure of the evaluated programs was estimated using the Memgen program. In **Paper III** however, the *bus_trans_mem.self* hardware counter is used to obtain the pressure that a program introduces into the system, which in turn improved the quality of the slowdown estimations. **Paper III** shows that most co-scheduled programs suffer a significant slowdown long before the aggregated bandwidth usage reaches the peak system bandwidth and that the correlation between the amounts of memory traffic a program generates and its sensitivity to memory contention is low.

Next, **Paper III** estimates the slowdown experienced by a subset of the NPB bench-marks [3] when using separate caches and a shared memory bus as well as when using a shared last-level cache and a shared memory bus. The slowdown predictions are then compared to co-scheduled measurements obtained using the brute-force method. The comparison shows that the predictions performed with separate caches are closer to the measured values than the shared cache predictions.

In **Paper IV** the full set of the NPB benchmarks [3] are analyzed, using the assumption that two processes are co-scheduled on each server. A total of 100 co-scheduling slowdowns were examined. **Paper IV** quantifies the differences between the slowdown predictions and brute-force slowdown measurements of co-scheduled program pairs. When comparing the predicted and measured slowdown values using regression analysis we find that the linear correlation coefficient (R^2) between the measured slowdowns and the predicted memory traffic based slowdowns are 0.890, which indicates a high correlation. The slowdown predictions systematically underestimate the slowdown compared to the brute-force measurements as it should. The measured slowdown is the sum of all slowdowns created by all shared resources while we only try to predict the slowdown caused by the sharing of the memory bus. Consequently, the average *predicted* slowdown is 5.40% while the average *measured* slowdown is 7.07%. To conclude, the slowdown prediction method presented in this section is able to perform a prediction of the slowdown co-scheduled programs experience when sharing the memory bus.

4 Memory Traffic Generation

The slowdown prediction method described in Section 3 relies on the ability to generate a stable memory bus load at custom load levels while having a minimal effect on the rest of the system. There are several programs that can be used to generate memory loads, the most famous being McCalpin's STREAM [17] which is used by the benchmarking suit Lmbench [34]. Since we wanted a simple but high performing traffic generator, the tool presented in **Paper II** generates memory traffic based on the Triad algorithm in STREAM [27].

The triad algorithm (Eq. 5) reads one value from memory and adds it to a register. It then adds a memory value to the same register and finally the new value is written back to memory at a third location. Thus, at a first glance, one iteration of the algorithm results in two read and one write operation, however in many modern systems with memory caching there is an additional read operation with intent to modify the value that will be overwritten (MESI: BusRdX [7]), which is performed before the write. The original Stream implementation does not take this extra read operation into account when calculating the bandwidth, but it is known by McCalpin [18]. To avoid cache hits the values are fetched from and written to the vectors *A*, *B*, and *C* which are several orders of magnitude larger than the last level cache.

$$c[j] = a[j] + b[j]$$

Eq. 5.

A pseudo code version of our assembler implementation of the triad algorithm is included in **Paper II**. To decrease the processor load, j is incremented in steps of eight instead of one as in the original Stream implementation, hence only one value is used in each cache line although all cache lines are fetched into the last level cache.

The structure of the memory generation and prediction tool is presented in **Paper II** and it follows the flowchart in Figure 2. The program is written in C, apart from the memory traffic generation kernel (described above) which is implemented in assembler. When Memgen is started the name of the program binary to evaluate is given as a command line argument. After performing some initial parsing and memory allocations, a second process is forked and locked to a specific core with the sched_setaffinity() system call. This new process then calls execv() to replace itself with the program binary to evaluate, which then starts to execute. The main process now forks one or several memory generator processes that lock themselves to different cores and generates memory traffic at the level specified on the command line. When the evaluated program finishes its execution the main process catches its return code and sends SIGINT to the memory generator processes. The runtime data is then post processed and the results are written to file.

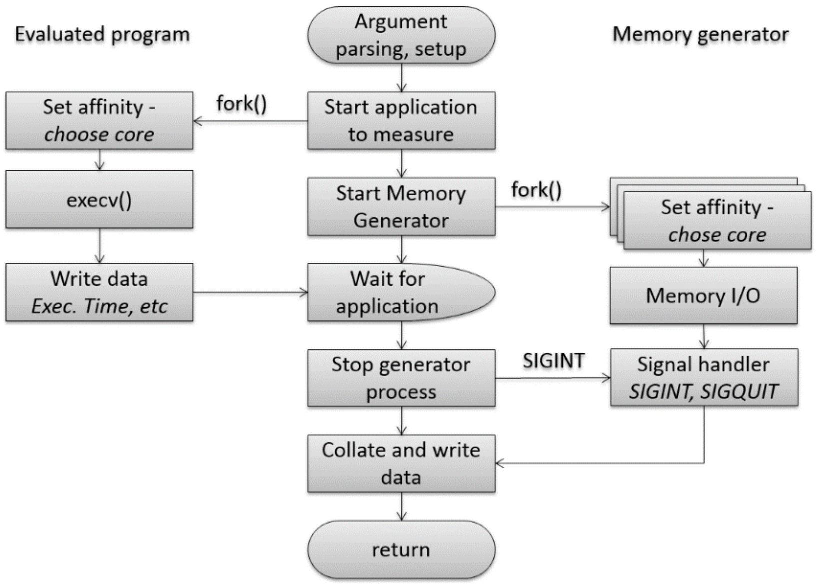

Figure 2. A flowchart of the evaluation process for each level of memory traffic.

Finally, **Paper II** evaluates Memgen’s ability to generate a specific and stable amount of memory traffic. In general, we find that the higher the amount of generated memory traffic is the higher are the fluctuations. Nevertheless, the highest memory throughput obtained by the STREM triad benchmark is 4921 MB/s and Memgen measured a maximum throughput of, on average, 4907 MB/s with a standard deviation of 1.12. During this test STREAM was modified to include the BusRdX read in its bandwidth calculation. When evaluating the execution of the memory generator together with processor- and memory bound benchmarks we find that when dealing with fully processor- or memory bound applications we can accurately predict the impact of executing two competing programs on the same multi-core machine.

Finally, **Paper II** illustrates how the Memgen traffic generator can be used to estimate the slowdown of two to four co-scheduled High Performance Linpack (HPL) [19] processes concurrently executing on the same quad-core computer. The execution time predictions in **Paper II** systematically overestimated the slowdown.

4.1 Related Work on Traffic Generation Tools

Since the introduction of the Memgen traffic generator in 2010, the Bubble-up [15] and the Bandwidth Bandit [8] traffic generators have also been presented. While the Memgen traffic generator constantly validates and invalidates different cache lines, the Bandwidth Bandit goes to great lengths to remove any cache contamination and only creates memory bus contention [8]. Thus, the bandit is able to generate memory traffic at varying rates (megabytes per second) without competing with other programs for cache resources.

Bubble-up on the other hand was designed to apply pressure on the memory subsystem as a whole and not only the memory bus [15]. In addition, Bubble-up is unaware of how much memory traffic it generates, it uses a third (reporter) process to calculate a value they refer to as a *bubble score* [16] which serves the same purpose as the *pressure* value described in Section 3. Bubble-up generates its traffic with the scalar kernel from Stream [17] as well as random loads and stores. The scalar kernel iterates over a memory area with an initial size of one megabyte, which is gradually increased. The initial size of one megabyte is small enough to fit in some L1 and most L2 caches, i.e. at the beginning, the accesses are local to the core Bubble-up is executing on.

Because of the memory footprint approach of Bubble-up it is unclear how much pressure in terms of memory bandwidth Bubble-up introduces into the system. As the bubble increases the characteristics of Bubble-up will change. At low levels Bubble-up will have large cache reuse (100%), thus aggressively claiming its cache space and effectively forcing competing processes' cache lines to be evicted. As the footprint increases the reuse frequency will decrease, allowing a co-running program to claim a relatively larger share of the caches. In [15], Mars et al. states that there is no one correct way to generate the traffic or estimate the pressure a program places on its co-running processes. However, the evaluation in Section 5 shows that the slowdown prediction method presented in this thesis is able to perform slowdown prediction that allows efficient co-scheduling.

5 Impact of Slowdown Prediction on Scheduling

Traditionally, cluster and cloud job-schedulers have scheduled one process to run on one server, which had one core. Only the really high end systems had multiple cores and they often only ran a handful of programs. However, with a plethora of different multi-core processors, memory speeds and systems running hundreds or thousands of different programs the basic scheduling problem becomes NP-complete. Memory contention aware scheduling has been offered as an effective strategy to mitigate the slowdown caused by the limited off-chip bandwidth [28]. In current research [1, 5, 10, 11, 21, 26, 28] several different characterization methods have been proposed as candidates to use as an input to resource aware scheduling to decrease the slowdown caused by the limited memory bus.

Blagodurov, et.al. [4] studied the relative performance of a set of cache based methods and identified that the memory controller, memory bus and prefetching hardware contributes more to the overall slowdown than caches do. Nevertheless, Fedorova, et.al. [9] reports that the last level cache (LLC) miss rate is an excellent method to use in an operating system scheduler in order to avoid memory based slowdown.

Several studies [1, 10, 13, 26] have reported on significant gains when performing co-scheduling allocations based on the processes memory bandwidth usage. According to Tang et.al. [22] a throughput increase of 40% is possible in datacenters when scheduling programs based on memory bandwidth usage.

To this end, **Paper IV** compares the trade-offs and relative effectiveness of four different methods aimed at enhancing the performance of memory resource aware scheduling. The four methods are the slowdown prediction method presented in Section 3, memory bandwidth usage, LLC miss rate, stack distance competition (SDC) [5]. **Paper IV** first explore the optimization space by simulating each and every possible way to schedule two instances of each of the ten NPB benchmarks in a setting of ten dual-core servers. The NPB benchmarks were executed for the same amount of time. This results in 1.44 million different schedules with a slowdown between 3.66% and 12.66%. The average throughput slowdown was 6.59%, see Figure 3 for the full bi-nominal distribution. For a method to make a positive contribution to the scheduling it has to perform better than the average slowdown. Furthermore, **Paper IV** shows that less than 0.01% or 98 schedules experience a slowdown of less than 4% and a mere 63,200 schedules has a slowdown of 5% or less.

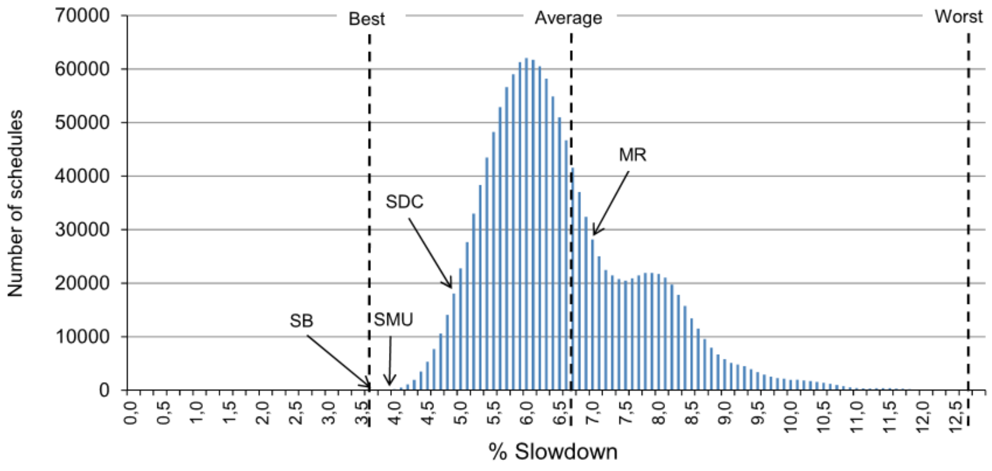

Figure 3. The throughput slowdown distribution of all possible ways to schedule 20 programs on ten dual-core computers. The best schedules according to the slowdown based (SB), memory bandwidth usage (MBU), stack distance competition (SDC), and last level cache miss rate (MR) methods are also presented as indicated by the arrows.

Paper **IV** shows that the schedule created by the slowdown prediction method is the 2nd best schedule with 3.69% slowdown. When scheduling using the memory bandwidth as input the slowdown is 3.94%, which makes it the 51st best schedule overall. The preferred schedule according to SDC has a slowdown of 4.94% and comes in the 52,446th place. One interesting finding in **Paper IV** is that the LLC miss rate results would increase the slowdown compared to a non-memory aware scheduler, which over time would converge towards the average slowdown.

Furthermore, **Paper IV** evaluates ten alternative scheduling scenarios using nine of the ten NPB benchmarks in each scenario. The results show that the slowdown prediction method, on average, generated schedules with an overall execution time only 0.44% slower than the best possible schedule, the corresponding number for non-memory aware scheduling is 2.89%. Furthermore, while the results show that using the slowdown, bandwidth or SDC methods will decreases the slowdown in all scenarios, the last level cache miss rate method performs worse than the average schedule in six of the eleven scenarios and using it will not impact system performance in either way.

6 Simple Scheme to Avoid Bad Co-Schedules

It is thus far assumed that a resource aware job-scheduler can increase the throughput of a cluster of cloud system by *selecting the best* ways in which processes can be scheduled. While this is true, the throughput can also be increased by *removing the worst* ways in which processes can be scheduled. Consider a job-scheduler that merely allocates one process to one core, totally disregarding any co-scheduling slowdowns. Over time the throughput will converge around the average slowdown, given an even distribution of execution time between all different programs. If the worst ways in which processes can be scheduled were removed the best throughput will still be the same, however the worst and the average throughputs will decrease.

To this end **Paper IV** have identified that the system throughput often suffers when a program is co-scheduled with another instance of itself. The experimental results from **Paper IV** and Section 5 show that seven of the ten co-schedules in the main scenario met this criteria. Furthermore, turning to the full set of scenarios the schedule where all programs are co-scheduled with another instance of itself was always one of the 15 worst. Thus in all eleven scenarios the schedule was among the worst 0.001% of schedules. The distribution of the two worst same instance co-schedules from **Paper IV** have been highlighted in Figure 4.

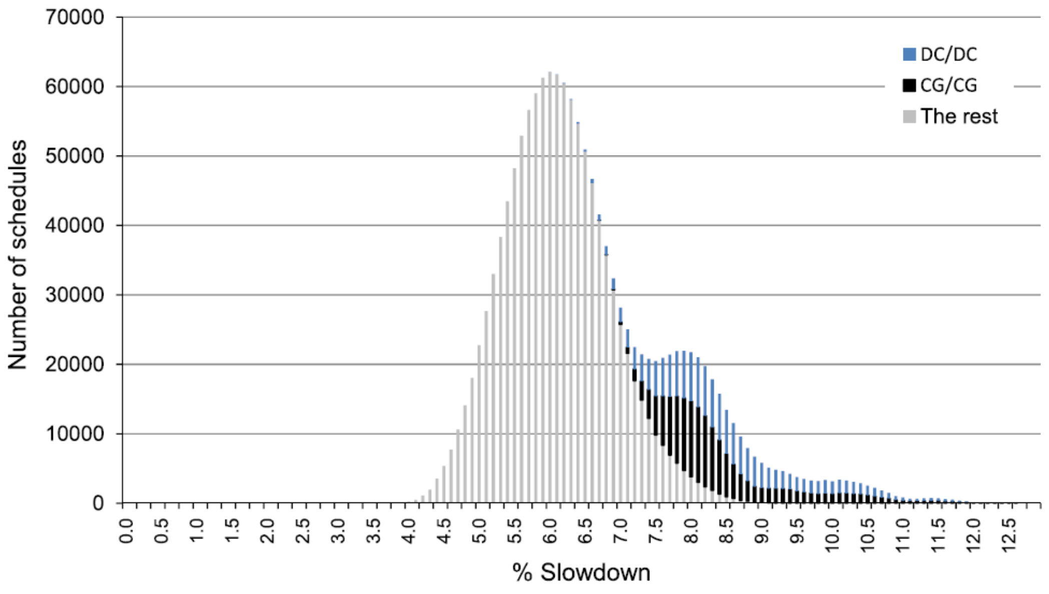

Figure 4. The schedules containing the two worst same instance co-schedules have been highlighted. The schedules including both co-schedules are included twice, once in DC/DC and once in CG/CG.

Conceptually, there are two major contributing factors: 1) Combining a program with a high degree of resource usage with another instance of itself will place a high load on that specific resource, thus slowing down both program executions. 2) Combining two programs with a low degree of resource usage removes the potential benefit of co-scheduling them with a program which uses said resource to a high degree.

Conclusively, **Paper IV** suggests that co-scheduling two instances of the same program is likely to decrease the overall system throughput, especially if the program has a high or low utilization of a scarce resource.

7 Concluding Remarks

The number and size of cluster and cloud systems are growing rapidly as they provide a computational platform for millions of users. Many industrial companies use in-house or outsourced systems to cater for their increasing demand for, among other things, engineering simulations. Nevertheless, in the background, much effort is put into improving the efficiency of said systems to meet current and future challenges. One of the main challenges is that the increasing level of on-chip parallelism combined with the limited off-chip bandwidth effectively turns the memory bus into a bottleneck. In this context, this thesis contributes with a novel slowdown prediction method which estimates the slowdown programs experience when executing concurrently on different cores in the same server. The method is motivated by the fact that for a job-scheduler to mitigate the slowdown caused by memory bus contention it must be able to, at least approximately, compare the impact of different scheduling decisions, or the scheduling becomes pure guesswork.

Paper I explores the impact resource sharing has on engineering simulation efficiency in an industrial dual-core cluster with regards to execution time as well as hardware and license costs. To this end, the paper proposes a cost metric, which in turn shows that the co-scheduling slowdown is a problem both from an execution time, as well as, a resource efficiency standpoint.

This thesis then focuses on predicting the slowdown caused by memory bus contention when programs are co-scheduled on the same server. **Paper III** presents a new slowdown prediction method which is able to, quite accurately, estimate the slowdown two programs experience when co-scheduled. The method is based on the hypothesis that if two programs, *A* and *B*, are co-scheduled the memory traffic generated by program *A* will cause program *B* to execute slower. First the slowdown prediction method determines the amount of memory traffic (*pressure*) a program introduces into the system. The method then determines how sensitive a program is to competing memory traffic by generating synthetic traffic while measuring how this affects the program's execution time. To this end, **Paper II** contributed with the Memgen tool which was used to generate stable streams of controlled traffic and measure the impact they had on the execution time of the investigated programs. Based on the slowdown values from five measurement points the program's sensitivity curve is interpolated. Hence, to determine how much program *A* will be slowed down by program *B*, the method looks up the slowdown value on *A's* sensitivity curve that corresponds to the level of memory traffic generated by *B*, and vice versa.

Paper IV shows that the method provides far better performance by selecting one of the best ways, although not the very best way, to schedule the processes with respect to the overall system throughput. Furthermore, **Paper IV** show that scheduling based on the slowdown prediction method yield a higher throughput than three other methods suggested for avoiding memory bandwidth slowdown.

The final contribution of **Paper IV** is that it identifies that the system throughput often suffers when a program is co-scheduled with another instance of itself. The paper investigated eleven scenarios and in all scenarios the schedule where all programs were co-scheduled with another instance of itself was among the overall worst schedules.

In this thesis the slowdown prediction method is evaluated using two co-scheduled processes on each multi-core server. The method, as presented in Section 3, can scale to handle multiple processes as long as each process has its own core. Future studies should evaluate the method's effectiveness when scheduling on more than two cores. Other, future questions are: what is the optimal scheduling policy to use in conjunction with slowdown prediction methods and how well will it perform in a real production environment. Finally, the results in this thesis are based on the slowdown prediction method and the Memgen tool. Although it outperforms comparable cache and memory bandwidth based methods it is so far unknown how well it performs compared to alternative traffic generation based approaches. However, the results show that the slowdown prediction method performs very well and it would be interesting to investigate if it can be enhanced by sharing ideas with other methods.

8 Bibliography

1. Antonopoulos C. D., Nikolopoulos D. S., Papatheodorou T. S., ”Realistic workload scheduling policies for taming the memory bandwidth bottleneck of smps”, *International Conference on High Performance Computing*, Springer-Verlag, Berlin, pp 286–296, 2004.

2. Becker D. J., Sterling T., Savarese D., Dorband J. E., Ranawak U. A., Packer C. V., "BEOWULF: A parallel workstation for scientific computation", *International Conference on Parallel Processing*, vol. 95, 1995.

3. Bailey D. H., "The NAS Parallel Benchmarks", *Encyclopedia of Parallel Computing*, Springer-Verlag, Berlin, 2009.

4. Blagodurov S., Zhuravlev S., Fedorova A., “Contention-aware scheduling on multicore systems”, *ACM Transactions on Computer Systems*, ACM, Vol. 28, New York, USA, 2010.

5. Chandra D., Guo F., Kim S., Solihin Y., ”Predicting inter-thread cache contention on a chip multi-processor architecture”, *International Symposium on High-Performance Computer Architecture*, IEEE Computer Society, Washington, DC, USA, pp 340–351, 2005.

6. Daci G., Tartari M., “A comparative review of contention-aware scheduling algorithms to avoid contention in multicore systems”, *Third international conference on trends in information, telecommunication and computing*, Springer, New York, pp 99–106, 2013.

7. Dubois M., Annavaram M., Stenström P., *Parallel Computer Organization and Design*, Cambridge University Press, 2012.

8. Eklov D., Nikoleris N., Black-Schaffer D., Hagersten E., ”Bandwidth bandit: Quantitative characterization of memory contention”, *International Conference on Parallel Architectures and Compilation Techniques*, ACM, New York, pp 457–458, 2012.

9. Fedorova A., Blagodurov S., Zhuravlev S., “Managing contention for shared resources on multicore processors”, *Communications of the ACM*, vol. 53, ACM, New York, USA, pp 49-57, 2010.

10. Field D., Johnson D., Mize D., Stober R., ”Scheduling to overcome the multi-core memory bandwidth bottleneck”, *Hewlett Packard and Platform Computing White Paper*, 2007.

11. Jiang Y., Shen X., Chen J., Tripathi R., ”Analysis and approximation of optimal co-scheduling on chip multiprocessors”, *International Conference on Parallel architectures and compilation techniques*, New York, USA, pp 220–229, 2008.

12. Koukis E., Koziris N., "Memory bandwidth aware scheduling for SMP cluster nodes", *Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp 187-196, 2005.

13. Koukis E., Koziris N., ”Memory and network bandwidth aware scheduling of multiprogrammed workloads on clusters of smps”, *International Conference on Parallel*

and Distributed Systems, Volume 1, IEEE Computer Society, Washington, DC, USA, pp 345–354, 2006.

14. Lindström P., de Blanche A., “Integration and Optimization of a 64-core HPC For FEM- and/or CFD Welding Simulations”, *Improving Simulation Prediction by Using Advanced Material Models*, NAFEMS NORDIC, Lund, Sweden, 2013.

15. Mars J., Tang L., Hundt R., Skadron K., Soffa ML., ”Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations”, *IEEE/ACM International Symposium on Microarchitecture*, ACM, New York, USA, 2011.

16. Mars J., Tang L., Hundt R., Skadron K., Soffa ML., ”Increasing utilization in warehouse scale computers using Bubble-up!”, *IEEE Micro*, 2012.

17. McCalpin J. D., ”Memory bandwidth and machine balance in current high performance computers”, *IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, pp 19–25, 1995.

18. McCalpin John D., “Answer to A Curious Bandwidth Result”, *https://software.intel.com/ en-us/forums/software-tuning-performance-optimization-platform-monitoring/ topic/559876*, 2015-06-12.

19. Petitet A., Whaley R. C., Dongarra J., Cleary A., ”HPL - a portable implementation of the high-performance Linpack benchmark for distributed-memory computers”, *http://www.netlib.org/benchmark/hpl/*, 2015.

20. Singer N., “More chip cores can mean slower supercomputing, sandia simulation shows”, *Sandia National Laboratories News Release*, 2009.

21. Tam D.K., Azimi R., Soares L.B., Stumm M., ”Rapidmrc: Approximating L2 miss rate curves on commodity systems for online optimizations”, *International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, New York, USA, pp 121–132, 2009.

22. Tang L., Mars J., Vachharajani N., Hundt R., Soffa ML., ”The impact of memory subsystem resource sharing on datacenter applications”, *International symposium on Computer Architecture*, ACM, New York, USA, pp 283–294, 2011.

23. Top 500 supercomputer site, *http://www.top500.org*, June, 2015.

24. Yotov K., Pingali K., Stodghill P., “Automatic measurement of memory hierarchy parameters.”, *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Banff, Alberta, Canada, June 06, 2005.

25. Wulf W. A., McKee S. A., “Hitting the memory wall: implications of the obvious.”, *SIGARCH Computer Architecture News* 23, pp 20-24, 1995.

26. Xu D., Wu C., Yew P.C., ”On mitigating memory bandwidth contention through bandwidth-aware scheduling”, *International Conference on Parallel architectures and compilation techniques*, New York, NY, USA, pp 237–248, 2010.

27. Zhuravlev S., Blagodurov S., Fedorova A., ”Addressing shared resource contention in multicore processors via scheduling”, *ASPLOS on Architectural support for programming languages and operating systems,* ACM, New York, USA, pp 129–142, 2010.

28. Zhuravlev S., Saez JC., Blagodurov S., Fedorova A., Prieto M., ”Survey of scheduling techniques for addressing shared resources in multicore processors”, *ACM Computer Surveys* 45(1), 2012.