Sven Karlsson - DTU Orbit (30/10/2019)

Sven Karlsson

Organisations

Department of Applied Mathematics and Computer Science
20/08/2018 → 06/09/2018 Former
svea@dtu.dk
VIP

Associate Professor, Department of Applied Mathematics and Computer Science
27/12/2012 → 06/09/2018 Former
svea@dtu.dk
VIP

Associate Professor, Department of Informatics and Mathematical Modeling
09/07/2007 → 07/04/2016 Former
ska@imm.dtu.dk
VIP

Embedded Systems Engineering
18/02/2013 → 06/09/2018 Former
VIP

Embedded Systems Engineering
25/02/2012 → 18/02/2013 Former
VIP

Research outputs:

Improving Loop Dependence Analysis
Programmers can no longer depend on new processors to have significantly improved single-thread performance. Instead, gains have to come from other sources such as the compiler and its optimization passes. Advanced passes make use of information on the dependencies related to loops. We improve the quality of that information by reusing the information given by the programmer for parallelization. We have implemented a prototype based on GCC into which we also add a new optimization pass. Our approach improves the amount of correctly classified dependencies resulting in 46% average improvement in single-thread performance for kernel benchmarks compared to GCC 6.1.

General information
Publication status: Published
Organisations: Formal Methods, Department of Applied Mathematics and Computer Science, Embedded Systems Engineering
Contributors: Jensen, N. B., Karlsson, S.
Number of pages: 24
Pages: 1-24
Publication date: 2017
Peer-reviewed: Yes

Publication information
Journal: ACM Transactions on Architecture and Code Optimization
Volume: 14
Issue number: 3
ISSN (Print): 1544-3566
Ratings:
BFI (2017): BFI-level 1
Scopus rating (2017): CiteScore 1.84 SJR 0.301 SNIP 1.01
Web of Science (2017): Impact factor 1.131
Web of Science (2017): Indexed yes
Original language: English
DOIs: 10.1145/3095754
Source: FindIt
A scalable lock-free hash table with open addressing

Concurrent data structures synchronized with locks do not scale well with the number of threads. As more scalable alternatives, concurrent data structures and algorithms based on widely available, however advanced, atomic operations have been proposed. These data structures allow for correct and concurrent operations without any locks. In this paper, we present a new fully lock-free open addressed hash table with a simpler design than prior published work. We split hash table insertions into two atomic phases: first inserting a value ignoring other concurrent operations, then in the second phase resolve any duplicate or conflicting values.

Our hash table has a constant and low memory usage that is less than existing lock-free hash tables at a fill level of 33% and above. The hash table exhibits good cache locality. Compared to prior art, our hash table results in 16% and 15% fewer L1 and L2 cache misses respectively, leading to 21% fewer memory stall cycles. Our experiments show that our hash table scales close to linearly with the number of threads and outperforms, in throughput, other lock-free hash tables by 19%.
A Scalable Prescriptive Parallel Debugging Model

Debugging is a critical step in the development of any parallel program. However, the traditional interactive debugging model, where users manually step through code and inspect their application, does not scale well even for current supercomputers due to its centralized nature. While lightweight debugging models, which have been proposed as an alternative, can currently only debug a subset of bug classes. We therefore propose a new model, which we call prescriptive debugging, to fill this gap between these two approaches. This user-guided model allows programmers to express and test their debugging intuition in a way that helps to reduce the error space. Based on this debugging model we introduce a prototype implementation embodying this model, the DySectAPI, allowing programmers to construct probe trees for automatic, event-driven debugging at scale. In this paper we introduce the concepts behind DySectAPI and, using both experimental results and analytical modelling, we show that the DySectAPI implementation can run with a low overhead on current systems. We achieve a logarithmic scaling of the prototype and show predictions that even for a large system the overhead of the prescriptive debugging model will be small.

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Language-Based Technology, Embedded Systems Engineering, Lawrence Livermore National Laboratory, Technical University of Denmark
Pages: 473-483
Publication date: 2015

Host publication information
Title of host publication: Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2015)
Publisher: IEEE
ISBN (Print): 978-1-4799-8648-4
DOIs:
10.1109/IPDPS.2015.15

Experiences with Compiler Support for Processors with Exposed Pipelines

Field programmable gate arrays, FPGAs, have become an attractive implementation technology for a broad range of computing systems. We recently proposed a processor architecture, Tinuso, which achieves high performance by moving complexity from hardware to the compiler tool chain. This means that the compiler tool chain must handle the increased complexity. However, it is not clear if current production compilers can successfully meet the strict constraints on instruction order and generate efficient object code. In this paper, we present our experiences developing a compiler backend using the GNU Compiler Collection, GCC. For a set of C benchmarks, we show that a Tinuso implementation with our GCC backend reaches a relative speedup of up to 1.73 over a similar Xilinx Micro Blaze configuration while using 30% fewer hardware resources. While our experiences are generally positive, we expose some limitations in GCC that need to be addressed to achieve the full performance potential of Tinuso.

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Language-Based Technology, Embedded Systems Engineering
Pages: 137-143
Publication date: 2015

Host publication information
Title of host publication: Proceedings of the 29th International Parallel and Distributed Processing Symposium Workshops (IPDPSW 2015)
Publisher: IEEE
ISBN (Print): 0-7695-5510-1
DOIs:
10.1109/IPDPSW.2015.9
Source: FindIt
Hardware Transactional Memory Optimization Guidelines, Applied to Ordered Maps

Synchronization of concurrent data structures is difficult to get right. Fine-grained synchronization locks small data chunks, but requires too high an overhead per chunk, traditional coarse-grained synchronization locks big data chunks, and thereby makes them unavailable to other threads. Neither synchronization method scales well. Recently, hardware transactional memory was introduced, which allows threads to use transactions instead of locks. So far, applying hardware transactional memory has shown mixed results. We believe this is because transactions are different from locks, and using them efficiently requires reasoning about those differences. In this paper we present 5 guidelines for applying hardware transactional memory efficiently, and apply the guidelines to BT-trees, a concurrent ordered map. Evaluating BT-trees on standard benchmarks shows that they are up to 5.3 times faster than traditional maps using hardware transactional memory, and up to 3.9 times faster than state of the art concurrent ordered maps.

A Synthesizable Multicore Platform for Microwave Imaging

Active microwave imaging techniques such as radar and tomography are used in a wide range of medical, industrial, scientific, and military applications. Microwave imaging devices emit radio waves and process their reflections to reconstruct an image. However, data processing remains a challenge as image reconstruction algorithms are computationally expensive and many applications come with strictly constrained mechanical or power requirements. We developed Tinuso, a multicore architecture optimized for performance when implemented on an FPGA. Tinuso’s architecture is well suited to run highly parallel image reconstruction applications at a low power budget. In this paper, we describe the design and the implementation of Tinuso’s communication structures, which include a generic 2D mesh on-chip interconnect and a network interface to the processor pipeline. We optimize the network for a latency of one cycle per network hop and attain a high clock frequency by pipelining the feedback loop to manage contention. We implement a multicore configuration with 48 cores and achieve a clock frequency as high as 300 MHz with a peak switching data rate of 9.6 Gbits/s per link on state-of-the-art FPGAs.
Automatic generation of application specific FPGA multicore accelerators

High performance computing systems make increasing use of hardware accelerators to improve performance and power properties. For large high-performance FPGAs to be successfully integrated in such computing systems, methods to raise the abstraction level of FPGA programming are required. In this paper we propose a tool flow, which automatically generates highly optimized hardware multicore systems based on parameters. Profiling feedback is used to adjust these parameters to improve performance and lower the power consumption. For an image processing application we show that our tools are able to identify optimal performance energy trade-offs points for a multicore based FPGA accelerator.

Code Commentary and Automatic Refactorings using Feedback from Multiple Compilers

Optimizing compilers are essential to the performance of parallel programs on multi-core systems. It is attractive to expose parallelism to the compiler letting it do the heavy lifting. Unfortunately, it is hard to write code that compilers are able to optimize aggressively and therefore tools exist that can guide programmers with refactorings allowing the compilers to optimize more aggressively. We target the problem with many false positives that these tools often generate, where the amount of feedback can be overwhelming for the programmer. Our approach is to use a filtering scheme based on feedback from multiple compilers and show how we are able to filter out 87.6% of the comments by only showing the most promising comments.
Compiler Feedback using Continuous Dynamic Compilation during Development

Optimizing compilers are vital for performance. However, compilers' ability to optimize aggressively is limited in some cases. To address this limitation, we have developed a compiler guiding the programmer in making small source code changes, potentially making the source code more amenable to optimization. This tool can help programmers understand what the optimizing compiler has done and suggest automatic source code changes in cases where the compiler refrains from optimizing. We have integrated our tool into an integrated development environment, interactively giving feedback as part of the programmers' development flow.

We have evaluated our preliminary implementation and show it can guide to a 12% improvement in performance. Furthermore, the tool can be used as an interactive optimization adviser improving the performance of the code generated by a production compiler. Here it can lead to a 153% improvement in performance, indicating the feasibility of the tool as a performance adviser for a production compiler.

DySectAPI: Scalable Prescriptive Debugging

We present the DySectAPI, a tool that allows users to construct probe trees for automatic, event-driven debugging at scale. The traditional, interactive debugging model, whereby users manually step through and inspect their application, does not scale well even for current supercomputers. While lightweight debugging models scale well, they can currently only debug a subset of bug classes. DySectAPI fills the gap between these two approaches with a novel user-guided approach. Using both experimental results and analytical modeling we show how DySectAPI scales and can run with a low overhead on current systems.
DySectAPI: Scalable Prescriptive Debugging

We present the DySectAPI, a tool that allows users to construct probe trees for automatic, event-driven debugging at scale. The traditional, interactive debugging model, whereby users manually step through and inspect their application, does not scale well even for current supercomputers. While lightweight debugging models scale well, they can currently only debug a subset of bug classes. DySectAPI fills the gap between these two approaches with a novel user-guided approach. Using both experimental results and analytical modeling we show how DySectAPI scales and can run with a low overhead on current systems.

ELB-trees - Efficient Lock-free B+trees

As computer systems scale in the number of processors, data structures with good parallel performance become increasingly important. Lock-free data structures promise improved parallel performance at the expense of higher complexity and sequential execution time. We present ELBtrees, a new lock-free dictionary with simple synchronization in the common case, making it almost 30 times faster than sequential library implementations at 24 threads.
Exposing MPI Objects for Debugging

Developers rely on debuggers to inspect application state. In applications that use MPI, the Message Passing Interface, the MPI runtime contains an important part of this state. The MPI Tools Working Group has proposed an interface for MPI Handle Introspection. It allows debuggers and MPI implementations to cooperate in extracting information from MPI objects. Information that can then be presented to the developer. MPI Handle Introspection provides a more general interface than previous work, such as Message Queue Dumping.

We add support for introspection to the TotalView debugger and a development version of Open MPI. We explain the interactions between the debugger and MPI library and demonstrate how MPI Handle Introspection raises the abstraction level to simplify debugging of MPI related programming errors.
Hardware Realization of an FPGA Processor - Operating System Call Offload and Experiences

Field-programmable gate arrays, FPGAs, are attractive implementation platforms for low-volume signal and image processing applications. The parallel structure of FPGAs allows for an efficient implementation of parallel algorithms. Sequential algorithms, on the other hand, often perform better on a microprocessor. It is therefore convenient for many applications to employ a synthesizable microprocessor to execute sequential tasks and custom hardware structures to accelerate parallel sections of an algorithm. In this paper, we discuss the hardware realization of Tinuso-I, a small synthesizable processor core that can be integrated in many signal and data processing platforms on FPGAs. We also show how we allow the processor to use operating system services. For a set of SPLASH-2 and SPEC2006 benchmarks we show an speedup of up to 64% over a similar Xilinx MicroBlaze implementation while using 27% to 35% fewer hardware resources.

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering
Contributors: Hindborg, A. E., Karlsson, S.
Number of pages: 4
Publication date: 2014
Peer-reviewed: No
Electronic versions:
poster
Source: PublicationPreSubmission
Source ID: 102241026
Research output: Contribution to conference › Poster – Annual report year: 2014 › Research

Hardware Realization of an FPGA Processor – Operating System Call Offload and Experiences

Field-programmable gate arrays, FPGAs, are attractive implementation platforms for low-volume signal and image processing applications.

The structure of FPGAs allows for an efficient implementation of parallel algorithms. Sequential algorithms, on the other hand, often perform better on a microprocessor. It is therefore convenient for many applications to employ a synthesizable microprocessor to execute sequential tasks and custom hardware structures to accelerate parallel sections of an algorithm. In this paper, we discuss the hardware realization of Tinuso-I, a small synthesizable processor core that can be integrated in many signal and data processing platforms on FPGAs. We also show how we allow the processor to use operating system services. For a set of SPLASH-2 and SPEC CPU2006 benchmarks we show a speedup of up to 64% over a similar Xilinx MicroBlaze implementation while using 27% to 35% fewer hardware resources.

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering, Language-Based Technology
Library Support for Resource Constrained Accelerators

Accelerators, and other resource constrained systems, are increasingly being used in computer systems. Accelerators provide power efficient performance and often provide a shared memory model. However, it is a challenge to map feature rich APIs, such as OpenMP, to resource constrained systems. In this paper, we present a lightweight system where an accelerator can remotely execute library functions on a host processor. The implementation takes up 750 bytes but can replace arbitrary library calls leading to significant savings in memory footprint. We evaluate with a set of SPLASH-2 applications and show that the impact on execution time is negligible when compared to GCCs OpenMP implementation.
MPI Debugging with Handle Introspection
The Message Passing Interface, MPI, is the standard programming model for high performance computing clusters. However, debugging applications on large scale clusters is difficult. The widely used Message Queue Dumping interface enables inspection of message queue state but there is no general interface for extracting information from MPI objects such as communicators. A developer can debug the MPI library as if it was part of the application, but this exposes an unneeded level of detail.

The Tools Working Group in the MPI Forum has proposed a specification for MPI Handle Introspection. It defines a standard interface that lets debuggers extract information from MPI objects. Extracted information is then presented to the developer, in a human readable format. The interface is designed to be independent of MPI implementations and debuggers.

In this paper, we describe our support for introspection in the TotalView debugger and test it against a reference introspection implementation in Open MPI. We also describe how the debugger interfaces with the MPI implementation.

General information
Publication status: Published
Contributors: Brock-Nannestad, L., DelSignore, J., Squyres, J. M., Karlsson, S., Mohror, K.
Number of pages: 4
Publication date: 2014
Peer-reviewed: Yes
Event: Paper presented at Workshop on Exascale MPI (ExaMPI 2014), New Orleans, United States.
Electronic versions:
exampi14_final.pdf
URLs:
https://www.pdc.kth.se/exampi14
Source: PublicationPreSubmission
Source ID: 104279024
Research output: Contribution to conference › Paper – Annual report year: 2014 › Research › peer-review

Safe Asynchronous System Calls
General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering
Contributors: Brock-Nannestad, L., Karlsson, S.
Number of pages: 1
Publication date: 2014
Peer-reviewed: Yes
Electronic versions:
cgo_poster_laub.pdf

Bibliographical note
Poster at the CGO2014 ACM Student Research Competition (SRC).
Source: PublicationPreSubmission
Source ID: 102283566
Research output: Contribution to conference › Poster – Annual report year: 2014 › Research › peer-review

Safe Asynchronous System Calls - extended abstract
General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering
Contributors: Brock-Nannestad, L., Karlsson, S.
Number of pages: 4
Publication date: 2014
Peer-reviewed: Yes
Electronic versions:
Bibliographical note
Extended abstract for poster at the CGO2014 ACM Student Research Competition (SRC).
Source: PublicationPreSubmission
Source ID: 102283572
Research output: Contribution to conference › Conference abstract for conference – Annual report year: 2014 › Research › peer-review

Smart Multicore Embedded Systems
This book provides a single-source reference to the state-of-the-art of high-level programming models and compilation tools for embedded system platforms. The authors address challenges faced by programmers developing software to implement parallel applications in embedded systems, where very often they are forced to rewrite sequential programs into parallel software, taking into account all the low level features and peculiarities of the underlying platforms. Readers will benefit from these authors' approach, which takes into account both the application requirements and the platform specificities of various embedded systems from different industries. Parallel programming tool-chains are described that take as input parameters both the application and the platform model, then determine relevant transformations and mapping decisions on the concrete platform, minimizing user intervention and hiding the difficulties related to the correct and efficient use of memory hierarchy and low level code generation.

Describes tools and programming models for multicore embedded systems
Emphasizes throughout performance per watt scalability
Discusses realistic limits of software parallelization
Enables software migration from single to multi-core
Includes coverage of fault-tolerance and dynamic reconfiguration
Uses case studies to demonstrate techniques presented

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering, University of Pisa, Delft University of Technology, French Alternative Energies and Atomic Energy Commission
Number of pages: 175
Publication date: 2014

Publication information
Publisher: Springer
ISBN (Print): 978-1-4614-8799-9
Original language: English
DOIs: 10.1007/978-1-4614-8800-2
Research output: Book/Report › Book – Annual report year: 2014 › Research › peer-review

Testing Infrastructure for Operating System Kernel Development
Testing is an important part of system development, and to test effectively we require knowledge of the internal state of the system under test. Testing an operating system kernel is a challenge as it is the operating system that typically provides access to this internal state information. Multi-core kernels pose an even greater challenge due to concurrency and their shared kernel state. In this paper, we present a testing framework that addresses these challenges by running the operating system in a virtual machine, and using virtual machine introspection to both communicate with the kernel and obtain information about the system. We have also developed an in-kernel testing API that we can use to develop a suite of unit tests in the kernel. We are using our framework for the development of our own multi-core research kernel.

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering
Contributors: Walter, M., Karlsson, S.
Number of pages: 4
Publication date: 2014

Host publication information
Title of host publication: Proceedings of the 7th Swedish Workshop on Multicore Computing (MCC’14)
Electronic versions: mcc_2014.pdf
Source: PublicationPreSubmission
Source ID: 104281963
Research output: Chapter in Book/Report/Conference proceeding › Article in proceedings – Annual report year: 2014 › Research › peer-review
ELB-trees an efficient and lock-free B-tree derivative

As computer systems scale in the number of processors, scalable data structures with good parallel performance become increasingly important. Lock-free data structures promise such improved parallel performance at the expense of higher algorithmic complexity and higher sequential execution time overhead. All lock-free data structures are based on simple atomic operations that, though supported by modern processors, are expensive in execution time. We present a lock-free data structure, ELB-trees, which under certain assumptions can be used as multimaps as well as priority queues. Specifically it cannot store duplicate key-value pairs, and it is not linearizable. Compared to existing data structures, ELB-trees require fewer atomic operations leading to improved performance. We measure the parallel performance of ELB-trees using a set of benchmarks and observe that ELB-trees are up to almost 30 times faster than library multimap implementations.
ELB-trees - Efficient Lock-free B+trees

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Language-Based Technology, Embedded Systems Engineering
Contributors: Bonnichsen, L. F., Karlsson, S., Probst, C. W.
Number of pages: 1
Publication date: 2013
Peer-reviewed: No
Electronic versions:
actualPoster_EBL_trees.pdf
Source: PublicationPreSubmission
Source ID: 102244024
Research output: Contribution to conference » Poster – Annual report year: 2014 » Research

Synthetic Aperture Radar Data Processing on an FPGA Multi-Core System
Synthetic aperture radar, SAR, is a high resolution imaging radar. The direct back-projection algorithm allows for a precise SAR output image reconstruction and can compensate for deviations in the flight track of airborne radars. Often graphic processing units, GPUs are used for data processing as the back-projection algorithm is computationally expensive and highly parallel. However, GPUs may not be an appropriate solution for applications with strictly constrained space and power requirements. In this paper, we describe how we map a SAR direct back-projection application to a multi-core system on an FPGA. The fabric consisting of 64 processor cores and 2D mesh interconnect utilizes 60% of the hardware resources of a Xilinx Virtex-7 device with 550 thousand logic cells and consumes about 10 watt. We apply software pipelining to hide memory latency and reduce the hardware footprint by 14%. We show that the system provides real-time processing of a SAR application that maps a 3000m wide area with a resolution of 2x2 meters.

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering, National Space Institute, Microwaves and Remote Sensing
Contributors: Schleuniger, P., Kusk, A., Dall, J., Karlsson, S.
Pages: 74-85
Publication date: 2013

Host publication information
Publisher: Springer
ISBN (Print): 978-3-642-36423-5
ISBN (Electronic): 78-3-642-36424-2
(Lecture Notes in Computer Science, Vol. 7767).
DOI: 10.1007/978-3-642-36424-2_7
Source: dtu
Source ID: u:6230
Research output: Chapter in Book/Report/Conference proceeding » Article in proceedings – Annual report year: 2013 » Research » peer-review

Task Based Programming on Embedded Multicores

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering
Contributors: Schleuniger, P., Karlsson, S.
Number of pages: 1
Publication date: 2013
Peer-reviewed: No
Event: Poster session presented at 8th International Conference on High-Performance and Embedded Architectures and Compilers, Berlin, Germany.
Electronic versions:
Design Principles for Synthesizable Processor Cores
As FPGAs get more competitive, synthesizable processor cores become an attractive choice for embedded computing. Currently popular commercial processor cores do not fully exploit current FPGA architectures. In this paper, we propose general design principles to increase instruction throughput on FPGA-based processor cores: first, superpipelining enables higher-frequency system clocks, and second, predicated instructions circumvent costly pipeline stalls due to branches. To evaluate their effects, we develop Tinuso, a processor architecture optimized for FPGA implementation. We demonstrate through the use of micro-benchmarks that our principles guide the design of a processor core that improves performance by an average of 38% over a similar Xilinx MicroBlaze configuration.

Guiding Programmers to Higher Memory Performance
Modern compilers use complex optimizations. It is often a problem for programmers to understand how source code should be written to enable optimizations. Interactive tools which guide programmers to higher performance are very important. We have developed such a tool that helps programmers modify their code to allow for aggressive optimization. In this paper, we extend it to support high level memory optimizations such as matrix reorganization. We evaluate the tool using two benchmarks and four different compilers. We show that it can guide the programmer to 22.9% higher performance.
Parallelizing More Loops with Compiler Guided Refactoring

The performance of many parallel applications relies not on instruction-level parallelism but on loop-level parallelism. Unfortunately, automatic parallelization of loops is a fragile process; many different obstacles affect or prevent it in practice. To address this predicament we developed an interactive compilation feedback system that guides programmers in iteratively modifying their application source code. This helps leverage the compiler’s ability to generate loop-parallel code. We employ our system to modify two sequential benchmarks dealing with image processing and edge detection, resulting in scalable parallelized code that runs up to 8.3 times faster on an eight-core Intel Xeon 5570 system and up to 12.5 times faster on a quad-core IBM POWER6 system. Benchmark performance varies significantly between the systems. This suggests that semi-automatic parallelization should be combined with target-specific optimizations. Furthermore, comparing the first benchmark to manually-parallelized, handoptimized pthreads and OpenMP versions, we find that code generated using our approach typically outperforms the pthreads code (within 93-339%). It also performs competitively against the OpenMP code (within 75-111%). The second benchmark outperforms manually-parallelized and optimized OpenMP code (within 109-242%).

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering, Chalmers University of Technology, IBM Research Laboratory
Pages: 410-419
Publication date: 2012

Host publication information
Title of host publication: 2012 41st International Conference on Parallel Processing (ICPP)
Publisher: IEEE
ISBN (Print): 978-1-4673-2508-0
(Conference Proceedings)
DOIs: 10.1109/ICPP.2012.48
Source: dtu
Source ID: u::7185

Software Managed Cache for Parallel Systems

General information
Publication status: Published
Organisations: Department of Applied Mathematics and Computer Science, Embedded Systems Engineering
Contributors: Schleuniger, P., Karlsson, S.
Number of pages: 1
Publication date: 2012
Peer-reviewed: No
Event: Poster session presented at 7th International Conference on High-Performance and Embedded Architectures and Compilers, Paris, France.
Electronic versions: hipeac12.pdf
Source: PublicationPreSubmission
Source ID: 104397794
Research output: Contribution to conference – Poster – Annual report year: 2014 – Research

Adapt or Become Extinct! The Case for a Unified Framework for Deployment-Time Optimization

The High-Performance Computing ecosystem consists of a large variety of execution platforms that demonstrate a wide diversity in hardware characteristics such as CPU architecture, memory organization, interconnection network, accelerators, etc. This environment also presents a number of hard boundaries (walls) for applications which limit software development (parallel programming wall), performance (memory wall, communication wall) and viability (power wall). The only way to survive in such a demanding environment is by adaptation. In this paper we discuss how dynamic information collected during the execution of an application can be utilized to adapt the execution context and may lead to performance gains beyond those provided by static information and compile-time adaptation. We consider specialization based on dynamic information like user input, architectural characteristics such as the memory hierarchy organization, and the execution profile of the application as obtained from the execution platform's performance monitoring units. One of the challenges of future execution platforms is to allow the seamless integration of these various kinds of information with information obtained from static analysis (either during ahead-of-time or just-in-time) compilation. We extend the notion of
information-driven adaptation and outline the architecture of an infrastructure designed to enable information ow and adaptation throughout the life-cycle of an application.

**General information**

Publication status: Published
Organisations: Embedded Systems Engineering, Department of Informatics and Mathematical Modeling, Language-Based Technology, National Technical University of Athens, Chalmers University of Technology, Swiss Federal Institute of Technology Zurich, National Research Center of High Performance Computers
Publication date: 2011

**Host publication information**

Title of host publication: EXADAPT ’11 Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era
Publisher: University of Strathclyde
ISBN (Print): 978-1-4503-0708-6
Electronic versions:
exadapt2011.pdf
DOIs:
10.1145/2000417.2000422
URLs:
http://exadapt.org/2011/

**Bibliographical note**

© ACM, 2011. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in EXADAPT ’11, http://doi.acm.org/10.1145/2000417.2000422
Source: orbit
Source ID: 313338
Research output: Chapter in Book/Report/Conference proceeding › Article in proceedings – Annual report year: 2011 › Research › peer-review

**Automatic Loop Parallelization via Compiler Guided Refactoring**

For many parallel applications, performance relies not on instruction-level parallelism, but on loop-level parallelism. Unfortunately, many modern applications are written in ways that obstruct automatic loop parallelization. Since we cannot identify sufficient parallelization opportunities for these codes in a static, off-line compiler, we developed an interactive compilation feedback system that guides the programmer in iteratively modifying application source, thereby improving the compiler’s ability to generate loop-parallel code. We use this compilation system to modify two sequential benchmarks, finding that the code parallelized in this way runs up to 8.3 times faster on an octo-core Intel Xeon 5570 system and up to 12.5 times faster on a quad-core IBM POWER6 system. Benchmark performance varies significantly between the systems. This suggests that semi-automatic parallelization should be combined with target-specific optimizations. Furthermore, comparing the first benchmark to hand-parallelized, hand-optimized pthreads and OpenMP versions, we find that code generated using our approach typically outperforms the pthreads code (within 93-339%). It also performs competitively against the OpenMP code (within 75-111%). The second benchmark outperforms hand-parallelized and optimized OpenMP code (within 109-242%).

**General information**

Publication status: Published
Organisations: Embedded Systems Engineering, Department of Informatics and Mathematical Modeling, IBM Research Laboratory, Chalmers University of Technology
Publication date: 2011

**Publication information**

Place of publication: Kgs. Lyngby, Denmark
Publisher: Technical University of Denmark
Original language: English
(IMM-Technical Report-2011; No. 12).
Electronic versions:
imm6041.pdf
Source: orbit
Source ID: 277862
Research output: Book/Report › Report – Annual report year: 2011 › Research
Comparing the Overhead of Lock-based and Lock-free Implementations of Priority Queues

With the advent of multi-core processors, concurrent data structures have received a renewed interest. While concurrent data structures where previously used mainly in high-performance computing, now they are found in all types of computer systems. A major challenge when designing such data structures is to allow multiple cores to safely access the data structure. Traditionally, mutual exclusion using lock primitives has been used to avoid interference between cores. However, lock primitives cause a high synchronization overhead in situations with high contention. More recently, lock-free data structures have been proposed as a solution to decrease the synchronization overhead. Lock-free data structures are more complex than their lock-based counterparts. It is not evident if the benefits of lower synchronization overhead can offset the higher sequential execution time caused by the complexity. In this paper, we compare a lock-free implementation of a priority queue with a lock-based implementation. We perform experiments with processors of different generations and observe large performance differences for lock-free data structures depending on the processor generation. The lock-free implementation performs much better on the most recent processor generation. We investigate this performance trend, using a set of micro-benchmarks and show a significant difference in the overhead of atomic operations between processor generations. The lock-free implementation executes approximately three times as many instructions as the lock-based implementation. However, the lock-free implementation outperforms the lock-based when multiple cores are used and the data structure is contended.

Compilation Driven Code Comments and Refactoring

Helping programmers write parallel software is an urgent problem given the popularity of multi-core architectures. Engineering compilers which automatically parallelize and vectorize code has turned out to be very challenging. Consequently, compilers are very selective with respect to the coding patterns they can optimize. We present an interactive approach and a tool set which leverages advanced compiler analysis and optimizations while retaining programmer control over the source code and its transformation. This allows optimization even when programmers refrain from enabling optimizations to preserve accurate debug information or to avoid bugs in the compiler. It also allows the source code to carry optimizations from one compiler to another. Secondly, our tool-set provides feedback on why optimizations do not apply to a code fragment and suggests workarounds which can be applied automatically. We demonstrate the ability of our tool to transform code, and suggest code refactoring that increase its amenability to optimization. The preliminary results shows that, with our tool-set, automatic loop parallelization with the GNU C compiler, gcc, yields 8.6x best-case speedup over the sequential version. The transformations and suggestions are based on the loop parallelization and matrix reorganization capabilities in the latest production release of gcc.

Efficient Co-Simulation of Multicore Systems

General information
Publication status: Published
Organisations: Embedded Systems Engineering, Department of Informatics and Mathematical Modeling
Contributors: Brock-Nannestad, L., Passas, S., Karlsson, S.
Publication date: 2011
Efficient Co-Simulation of Multicore Systems
Simulation is an indispensable tool for debugging and verification of multicore systems. However, simulation is slow. For complex multicore systems, a simulation model will execute several orders of magnitude slower than the actual hardware implementation. We propose a method for capturing the hardware state of a multicore design while it is running on an FPGA. With minimal changes to the design and using only the built-in JTAG programming and debugging facilities, we describe how to transfer the state from an FPGA to a simulator. We also show how the state can be transferred back from the simulator to FPGA. Given that the design runs in real-time on the FPGA, the end result is speed improvements of orders of magnitude over traditional pure software simulation.

Expressing Coarse-Grain Dependencies Among Tasks in Shared Memory Programs
Designers of embedded systems face tight constraints on resources, response time and cost. The ability to analyze embedded systems is essential to timely delivery of new designs. Many analysis techniques model parallel programs as task graphs. Task graphs capture the worst-case execution times of individual program tasks and the data dependencies among these. This paper introduces two compiler directives which let programmers annotate source code with data dependencies among tasks. Compiler analysis overapproximates the actual dependencies among tasks. The directives help eliminate potential data dependencies that do not occur at runtime. The directives cannot be verified at compile time. Therefore, the check for correct use is done at runtime—not unlike dynamic array bounds checking in many languages. The overhead of verifying the correct use of the directives was measured on a set of benchmarks on two platforms. The overhead of runtime checks was found to be negligible in all instances.
Hardware Support for Dynamic Languages

In recent years, dynamic programming languages have enjoyed increasing popularity. For example, JavaScript has become one of the most popular programming languages on the web. As the complexity of web applications is growing, compute-intensive workloads are increasingly handed off to the client side. While a lot of effort is put in increasing the performance of web browsers, we aim for multicore systems with dedicated cores to effectively support dynamic languages. We have designed Tinuso, a highly flexible core for experimentation that is optimized for high performance when implemented on FPGA. We composed a scalable multicore configuration where we study how hardware support for software speculation can be used to increase the performance of dynamic languages.

General information
Publication status: Published
Organisations: Embedded Systems Engineering, Department of Informatics and Mathematical Modeling, Language-Based Technology
Contributors: Schleuniger, P., Karlsson, S., Probst, C. W.
Publication date: 2011

Host publication information
Title of host publication: ACACES 2011 Seventh International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems
Electronic versions:
acaces_abstract_pass.pdf
URLs:
http://www.hipeac.net/summerschool/
Source: orbit
Source ID: 314922
Research output: Chapter in Book/Report/Conference proceeding › Article in proceedings – Annual report year: 2011 › Research › peer-review

Hardware Support for Dynamic Languages

General information
Publication status: Published
Organisations: Embedded Systems Engineering, Department of Informatics and Mathematical Modeling, Language-Based Technology
Contributors: Schleuniger, P., Karlsson, S., Probst, C. W.
Publication date: 2011
Peer-reviewed: Yes
Event: Poster session presented at 7th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems, Fiuggi, Italy.
Electronic versions:
acaces_poster_pass.pdf
URLs:
http://www.hipeac.net/summerschool/
Source: orbit
Source ID: 314927
Research output: Contribution to conference › Poster – Annual report year: 2011 › Research › peer-review

Improving Performance of Software Implemented Floating Point Addition

General information
Improving Performance of Software Implemented Floating Point Addition

We outline and evaluate hardware extensions to an integer processor pipeline which allow IEEE 754 floating point, FP, addition to be efficiently implemented in software. With a very moderate increase in hardware resources, our performance evaluation shows that, for a benchmark that executes 12.5% FP addition instructions, our approach exhibits a relative slowdown of 3.38 to 15.15 as compared to dedicated hardware. This is a significant improvement of pure software emulation which leads to relative slowdowns up to 45.33.

SRC: FenixOS - A Research Operating System Focused on High Scalability and Reliability

Computer systems keep increasing in size. Systems scale in the number of processing units, memories and peripheral devices. This creates many and diverse architectural trade-offs that the existing operating systems are not able to address. We are designing and implementing, FenixOS, a new operating system that aims to improve the state of the art in scalability and reliability. We achieve scalability through limiting data sharing when possible, and through extensive use of lock-free data structures. Reliability is addressed with a careful re-design of the programming interface and structure of the operating system.
Towards a Time-predictable Dual-Issue Microprocessor: The Patmos Approach

Current processors are optimized for average case performance, often leading to a high worst-case execution time (WCET). Many architectural features that increase the average case performance are hard to be modeled for the WCET analysis. In this paper we present Patmos, a processor optimized for low WCET bounds rather than high average case performance. Patmos is a dual-issue, statically scheduled RISC processor. The instruction cache is organized as a method cache and the data cache is organized as a split cache in order to simplify the cache WCET analysis. To fill the dual-issue pipeline with enough useful instructions, Patmos relies on a customized compiler. The compiler also plays a central role in optimizing the application for the WCET instead of average case performance.

Compiler Driven Code Comments and Refactoring

Helping programmers write parallel software is an urgent problem given the popularity of multi-core architectures. Engineering compilers which automatically parallelize and vectorize code has turned out to be very challenging and consequently compilers are very selective with respect to the coding patterns they can optimize. We present an interactive approach which leverages advanced compiler analysis and optimizations while retaining programmer control over the source code and its transformation. This allows optimization even when programmers refrain from enabling optimizations to preserve accurate debug information or to avoid bugs in the compiler. It also allows the source code to carry optimizations from one compiler to another. Secondly, our tool-set provides feedback on why optimizations do not apply to a code fragment and suggests workarounds which can be applied automatically. We demonstrate the ability of our tool to transform code, and suggest code refactoring that increases the amenability to optimization. The transformations and suggestions are based on the loop parallelization and matrix reorganization capabilities in the GNU Compiler Collection.
Expressing Inter-task Dependencies between Parallel Stencil Operations

Complex embedded systems are designed under tight constraints on response time, resource usage and cost. Design space exploration tools help designers map and schedule embedded software to complex architectures such as heterogeneous MPSoC's. Task graphs are coarse grained representations of parallel program behaviour which are used to evaluate the feasibility of a particular design. However, automatically extracting an accurate task graph from source code is challenging. This paper investigates how to describe data dependencies to aid tools based on program analysis in extracting task graphs from source code. We will examine a common parallel programming pattern - stencil operations - and show that even for such codes with a regular control flow, the precise dependencies between two stencil operations cannot always be determined at compile time. We introduce a language construct which i) captures an upper bound on the number of dependencies between successive stencil operations and ii) instructs the compiler to generate code which ensures that the bound holds for each execution of the program. The impact of our proposal is evaluated using a micro-benchmark and two soft real-time embedded image processing applications. The coding effort is low - at most one line of code per parallel loop was added. The performance impact is evaluated on a quad-core Linux workstation and we observe no statistically significant slowdown.

High Performance Triangle versus Box Intersection Checks

Tinuso: A processor architecture for a multi-core hardware simulation platform

Multi-core systems have the potential to improve performance, energy and cost properties of embedded systems but also require new design methods and tools to take advantage of the new architectures. Due to the limited accuracy and performance of pure software simulators, we are working on a cycle accurate hardware simulation platform. We have developed the Tinuso processor architecture for this platform. Tinuso is a processor architecture optimized for FPGA implementation. The instruction set makes use of predicated instructions and supports C/C++ and assembly language programming. It is designed to be easy extendable to maintain the exibility required for the research on multi-core systems. Tinuso contains a co-processor interface to connect to a network interface. This interface allow for communication over an on-chip network. A clock frequency estimation study on a deeply pipelined Tinuso implementation shows that it is feasible to operate it at a frequency higher than 300 MHz in current state-of-the-art high end FPGAs.
Adaptable Support for Programming Models in Many-core Architectures

General information
Publication status: Published
Organisations: Embedded Systems Engineering, Department of Informatics and Mathematical Modeling
Contributors: Rasmussen, M. S., Karlsson, S., Sparsø, J.
Number of pages: 22
Publication date: 2009

Host publication information
Title of host publication: Proceeding of third swedish workshop on multi-core computing - MCC'10
Volume: 3
Source: orbit
Source ID: 271130
Research output: Chapter in Book/Report/Conference proceeding › Article in proceedings – Annual report year: 2010 › Research

Identifying Inter-task Communication in Shared Memory Programming Models

Modern computers often use multi-core architectures, covering clusters of homogeneous cores for high performance computing, to heterogeneous architectures typically found in embedded systems. To efficiently program such architectures, it is important to be able to partition and map programs onto the cores of the architecture. We believe that communication patterns need to become explicit in the source code to make it easier to analyze and partition parallel programs. Extraction of these patterns are difficult to automate due to limitations in compiler techniques when determining the effects of pointers. In this paper, we propose an OpenMP extension which allows programmers to explicitly declare the pointer based data-sharing between coarse-grain program parts. We present a dependency directive, expressing the input and output relation between program parts and pointers to shared data, as well as a set of runtime operations which are necessary to enforce declarations made by the programmer. The cost and scalability of the runtime operations are evaluated using micro-benchmarks and a benchmark from the NAS parallel benchmark suite. The measurements show that the overhead of the runtime operations is small. In fact, no performance degradation is found when using the runtime operations in the benchmark from the NAS parallel benchmark suite.

General information
Publication status: Published
Organisations: Embedded Systems Engineering, Department of Informatics and Mathematical Modeling
Contributors: Larsen, P., Karlsson, S., Madsen, J.
Number of pages: 183
Pages: 168-182
Publication date: 2009

Host publication information
Title of host publication: Lecture Notes in Computer Science : Evolving OpenMP in an Age of Extreme Parallelism
Publisher: Springer Berlin / Heidelberg
ISBN (Print): 978-3-642-02284-5
(Lecture Notes in Computer Science; No. 5568).
DOIs: 10.1007/978-3-642-02303-3
URLs: http://www.springerlink.com/content/675360146i42p5i2/?p=2f5f11a9fee84f03b0230c852dfe416&pi=13
Source: orbit
Source ID: 252071
Research output: Chapter in Book/Report/Conference proceeding › Conference abstract in proceedings – Annual report year: 2009 › Research › peer-review

Parallelism and Scalability in an Image Processing Application

The recent trends in processor architecture show that parallel processing is moving into new areas of computing in the form of many-core desktop processors and multi-processor system-on-chips. This means that parallel processing is
required in application areas that traditionally have not used parallel programs. This paper investigates parallelism and scalability of an embedded image processing application. The major challenges faced when parallelizing the application were to extract enough parallelism from the application and to reduce load imbalance. The application has limited immediately available parallelism and further extraction of parallelism is limited by small data sets and a relatively high parallelization overhead. Load balance is difficult to obtain due to the limited parallelism and made worse by non-uniform memory latency. Three parallel OpenMP implementations of the application are discussed and evaluated. We show that with some modifications relative speedups in excess of 9 on a 16 CPU system can be reached.

Performance Analysis of a Hardware/Software-based Cache Coherence Protocol in Shared Memory MPSoCs

In this work we examine the implications of building a single logical link out of multiple physical links. We use MultiEdge to examine the throughput-CPU utilization tradeoffs and examine how overheads and performance scale with the number and speed of links. We use low-level instrumentation to understand associated overheads, we experiment with setups between 1 and 8 1-GBit/s links, and we contrast our results with a single 10-GBit/s link. We find that: (a) Our base protocol achieves up-to 65% of the nominal aggregate throughput, (b) Replacing the interrupts with polling significantly impacts only the multiple link configurations, reaching 80% of nominal throughput, (c) The impact of copying on CPU overhead is significant, and removing copying results in up-to 66% improvement in maximum throughput, reaching almost 100% of the nominal throughput, (d) Scheduling packets over heterogeneous links requires simple but dynamic scheduling to account for different link speeds and varying load.

Exploiting spatial parallelism in Ethernet-based cluster interconnects.

In this work we examine the implications of building a single logical link out of multiple physical links. We use MultiEdge to examine the throughput-CPU utilization tradeoffs and examine how overheads and performance scale with the number and speed of links. We use low-level instrumentation to understand associated overheads, we experiment with setups between 1 and 8 1-GBit/s links, and we contrast our results with a single 10-GBit/s link. We find that: (a) Our base protocol achieves up-to 65% of the nominal aggregate throughput, (b) Replacing the interrupts with polling significantly impacts only the multiple link configurations, reaching 80% of nominal throughput, (c) The impact of copying on CPU overhead is significant, and removing copying results in up-to 66% improvement in maximum throughput, reaching almost 100% of the nominal throughput, (d) Scheduling packets over heterogeneous links requires simple but dynamic scheduling to account for different link speeds and varying load.
**Parallelism and Scalability in an Image Processing Application**

The recent trends in processor architecture show that parallel processing is moving into new areas of computing in the form of many-core desktop processors and multi-processor system-on-chip. This means that parallel processing is required in application areas that traditionally have not used parallel programs. This paper investigates parallelism and scalability of an embedded image processing application. The major challenges faced when parallelizing the application were to extract enough parallelism from the application and to reduce load imbalance. The application has limited immediately available parallelism. It is difficult to further extract parallelism since the application has small data sets and parallelization overhead is relatively high. There is also a fair amount of load imbalance which is made worse by a non-uniform memory latency. Even so, we show that with some tuning relative speedups in excess of 9 on a 16 CPU system can be reached.

**MultiEdge: An Edge-based Communication Subsystem for Scalable Commodity Servers**

At the core of contemporary high performance computer systems is the communication infrastructure. For this reason, there has been a lot of work on providing low-latency, high-bandwidth communication subsystems for clusters. In this paper, we introduce MultiEdge, a connection oriented communication system designed for high-speed commodity hardware. MultiEdge provides support for end-to-end flow-control, ordering, and reliable transmission. It transparently supports multiple physical links within a single connection. We use MultiEdge to examine the behavior of edge-based protocols using both micro-benchmarks and real-life shared memory applications. Our results show that MultiEdge is able to deliver about 88% of the nominal link throughput with a single 10-GBit/s link and more than 95% with multiple 1-GBit/s links. Our application results show that performing all of the communication protocol at the edge does not seem to cause any degradation in performance.
Projects:

**Systemarkitekturen baseret på Network-on-Chip**
Rasmussen, M. S., PhD Student, Department of Informatics and Mathematical Modeling
Sparsø, J., Main Supervisor
Karlsson, S., Supervisor
Madsen, J., Supervisor
Probst, C. W., Examiner
Grahn, H., Examiner
Nurmi, J. A., Examiner
DTU stipendium
01/10/2006 → 29/09/2010
Award relations: Systemarkitekturen baseret på Network-on-Chip
Project: PhD

**Test-driven model-based development of complex embedded systems**
Munck, A., PhD Student, Department of Mathematics
Madsen, J., Main Supervisor
Lindqvist, L., Supervisor
Mortensen, R., Supervisor
Pop, P., Supervisor
Karlsson, S., Examiner
Nielsen, P. Ø., Examiner
Larsen, P. G., Examiner
Industrial PhD
01/08/2013 → 16/08/2017
Award relations: Test-driven model-based development of complex embedded systems
Project: PhD

**Cognitive and Perceptive Cameras - Compilation system**
Jensen, N. B., PhD Student, Department of Mathematics
Probst, C. W., Main Supervisor
Karlsson, S., Supervisor
Lluch Lafuente, A., Examiner
Kessler, C. W., Examiner
Sestoft, P., Examiner
1/3 FUU, 1/3 inst 1/3 Andet
01/10/2013 → 18/01/2017
Award relations: Cognitive and Perceptive Cameras - Compilation system
Project: PhD

**Portable and Predictable Performance Heterogeneous Embedded Manycores - Upper Level System stack**
Bonnichsen, L. F., PhD Student, Department of Mathematics
Probst, C. W., Main Supervisor
Karlsson, S., Supervisor
Lluch Lafuente, A., Examiner
Hansen, R. R., Examiner
Assmann, U., Examiner
1/3 FUU, 1/3 inst 1/3 Andet
01/10/2012 → 21/01/2016
Award relations: Portable and Predictable Performance Heterogeneous Embedded Manycores - Upper Level System stack
Project: PhD
Tools for model-based software descriptions
Larsen, P., PhD Student, Department of Informatics and Mathematical Modeling
Karlsson, S., Main Supervisor
Madsen, J., Supervisor
Probst, C. W., Examiner
Cohen, A., Examiner
Stenström, P., Examiner
DTU stipendium
01/02/2007 → 31/08/2011
Award relations: Tools for model-based software descriptions
Project: PhD

Architecture concepts for improving programmability of multi-core architectures
Schleuniger, P., PhD Student, Department of Mathematics
Karlsson, S., Main Supervisor
Madsen, J., Supervisor
Pop, P., Examiner
Grahn, H., Examiner
Ramirez, A., Examiner
Forskningsrådsfinansiering
15/06/2010 → 26/05/2014
Award relations: Architecture concepts for improving programmability of multi-core architectures
Project: PhD

Activities:

HIPEAC - European Network of Excellence on High Performance and Embedded Architecture and Compilation (External organisation)
Period: 27 Jan 2008 → 31 Dec 2018
Sven Karlsson (Participant)
Department of Applied Mathematics and Computer Science
Embedded Systems Engineering
Degree of recognition: International

Related external organisation

European Network of Excellence on High Performance and Embedded Architecture and Compilation
Activity: Membership › Membership of research networks or expert groups