

Automatic Application Tuning for HPC Architectures

Edited by

Siegfried Benkner¹, Franz Franchetti², Hans Michael Gerndt³, and Jeffrey K. Hollingsworth⁴

1 Universität Wien, AT, siegfried.benkner@univie.ac.at

2 Carnegie Mellon University – Pittsburgh, US, franzf@ece.cmu.edu

3 TU München, DE, gerndt@in.tum.de

4 University of Maryland – College Park, US, hollings@cs.umd.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 13401 “Automatic Application Tuning for HPC Architectures”. This workshop featured a series of talks and four breakout sessions on hot topics in the area of autotuning. The field of autotuning for HPC applications is of growing interest and many research groups around the world are currently involved. However, the field is still rapidly evolving with many different approaches being taken to autotuning. This workshop provided an opportunity to discuss these many approaches, and help to unify the terminology used by different groups.

Seminar 29. September to 4. October, 2014 – www.dagstuhl.de/13401

1998 ACM Subject Classification D.2 Software Engineering, D.2.1 Requirements/Specifications, D.2.8 Metrics, D.2.11 Software Architectures

Keywords and phrases Parallel Computing, Programming Tools, Performance Analysis and Tuning

Digital Object Identifier 10.4230/DagRep.3.9.214

1 Executive Summary

Siegfried Benkner

Franz Franchetti

Hans Michael Gerndt

Jeffrey K. Hollingsworth

License © Creative Commons BY 3.0 Unported license
© Siegfried Benkner, Franz Franchetti, Hans Michael Gerndt, and Jeffrey K. Hollingsworth

Parallel computer systems especially for High Performance Computing are getting increasingly complex. The reasons are manifold. HPC systems today with a peak performance of several petaflops have hundreds of thousands of cores that have to be able to work together efficiently. Those machines have a deep hierarchy, which has to be understood by the programmer to tune his program so that it profits from higher interconnection rates. In addition, to reduce the power consumption of those systems, advanced hard- and software techniques are applied, such as the usage of GPUs that are highly specialized for regular data parallel computations via simple compute cores and high bandwidth to the graphics memory. Another technique is to reduce the clock frequency of processors when appropriate, e.g. when the application or phases of the execution are memory bound. This transforms a homogeneous system into



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Automatic Application Tuning for HPC Architectures, *Dagstuhl Reports*, Vol. 3, Issue 9, pp. 214–244

Editors: Siegfried Benkner, Franz Franchetti, Hans Michael Gerndt, and Jeffrey K. Hollingsworth



DAGSTUHL REPORTS

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a heterogeneous system, which complicates programming tasks such as load balancing and efficient communication.

The complexity of today's parallel architectures has a significant impact on the performance of parallel applications. Due to the high amount of energy and money being lost because of the low processor utilization, application developers are now investing significant time to tune their codes for the current and emerging systems. This tuning is a cyclic process of gathering data, identifying code regions that can be improved, and tuning those code regions.

There are a growing number of autotuning researchers in Europe, the United States, and Asia. However, there are relatively few opportunities for these researchers to meet together. The unique format of a Dagstuhl seminar provides the opportunity to bring together researchers from around the world that are using different approaches to autotuning.

This workshop brought together those people working on autotuning with people working on performance analysis tools. While the analysis tools indicate performance problems, their combination with performance tuning might make those tools even more successful. The presentations of experts in both areas will increase the interest and the knowledge of the techniques applied in the other area. It will steer future collaborations and might also lead to concrete ideas for coupling performance analysis and performance tuning tools.

The workshop was driven by the European FP7 project AutoTune that started on October 15th, 2011. It is the goal of AutoTune to implement the Periscope Tuning Framework based on the automatic performance analysis tool Periscope. It will couple Periscope's performance analysis with performance and energy efficiency tuning in an online approach.

Performance Analysis. Performance analysis tools support the programmer in the first two tasks of the tuning cycle. Performance data are gathered during program execution by monitoring the application's execution. Performance data are both summarized and stored as profile data or all details are stored in so called trace files. In addition to application monitoring, performance analysis tools also provide means to analyze and interpret the provided performance data and thus to detect performance problems. The analysis is either supported by graphical display or by annotating the source code.

State of the art performance analysis tools fall into two major classes depending on their monitoring approach: profiling tools and tracing tools. Profiling tools summarize performance data for the overall execution and provide information such as the execution time for code regions, number of cache misses, time spent in MPI routines, and synchronization overhead for OpenMP synchronization constructs. Tracing tools provide information about individual events, generate typically huge trace files and provide means to visually analyze those data to identify bottlenecks in the execution.

Representatives for these two classes are gprof, OMP and Vampir. Gprof is the GNU Profiler tool. It provides a flat profile and a callpath profile for the program's functions. The measurements are done by instrumenting the application. OmpP is a profiling tool for OpenMP developed at TUM and the University of Tennessee. It is based on instrumentation with Opari and determines certain overhead categories of parallel regions. In contrast to the previous two tools, Vampir is a commercial trace-based performance analysis tool from Technische Universität Dresden. It provides a powerful visualization of traces and scales to thousands of processors based on a parallel visualization server.

The major research challenges in the development of PA tools are to automate the analysis and to improve the scalability of the tools. Automation of the analysis is important to facilitate the application developer's task. Starting from the formalization of performance properties in the European-American working group APART (<http://www.fz-juelich.de/apart>), automatic performance analysis tools were developed. Paradyn from University of Wisconsin was the first

automatic online analysis tool. Its performance consultant guided the search for performance bottlenecks while the application was executing. The most important representatives are SCALASCA and Periscope. SCALASCA is an automatic performance analysis tool developed at Forschungszentrum Jülich and the German Research School on Simulation Sciences. It is based on performance profiles as well as on traces. The automatic trace analysis determines MPI wait time via a parallel trace replay on the application's processors after the application execution terminated.

Periscope is an automatic performance analysis tool for highly parallel applications written in MPI and/or OpenMP currently under development at Technische Universität München. It is a representative for a class of automatic performance analysis tools automating the whole analysis procedure. Unique to Periscope is that it is an online tool and it works in a distributed fashion. This means that the analysis is done while the application is executing (online) and by a set of analysis agents, each searching for performance problems in a subset of the application's processes (distributed). The properties found by Periscope point to code regions that might benefit from further tuning.

Performance Autotuning. The central part of the tuning process is the search for the best combination of code transformations and parameter settings of the execution environment. This creates an enormous search space, which further complicates the whole tuning task. As a result, much research has been dedicated to the area of autotuning in the last years and many different ideas have been gathered. These can be grouped into four categories:

- self-tuning libraries for linear algebra and signal processing like ATLAS, FFTW, OSKI and SPIRAL;
- tools that automatically analyze alternative compiler optimizations and search for their optimal combination;
- autotuners that search a space of application-level parameters that are believed to impact the performance of an application;
- frameworks that try to combine ideas from all the other groups.

The first category contains special purpose libraries that are highly optimized for one specific area. The Automatically Tuned Linear Algebra Software (ATLAS) supports the developers in creating numerical programs. It automatically generates and optimizes the popular Basic Linear Algebra Subroutines (BLAS) kernels for the currently used architecture. Similarly, FFTW is a library for computing the discrete Fourier transform on different systems. Due to the FFTW design, an application using it will perform well on most architectures without modification.

However, the growing diversity of parallel application areas requires a more general autotuning strategy. Thus, substantial research has been done in a different application-independent approach of autotuning. This is based on the automatic search for the right compiler optimizations on the specific platform. Such tools can be separated into two groups according to their methodology: iterative search tools and those using machine learning techniques. There has been much work in the first category. All these tools share the idea of iteratively enabling certain optimizations. They run the compiled program and monitor its performance. Based on the outcome, they decide on the new tuning combination. Due to the huge size of the search space, these tools are relatively slow. There exists an algorithm called combined elimination (CE) that greatly improves the previous search-based methods.

The second branch of compiler-based autotuners applies a different strategy to look for the best optimization settings. They use knowledge about the program's behavior and machine learning techniques to select the optimal combination. This approach is based

on an automatically built per-system model, which maps performance counters to good optimization options. This model can then be used with different applications to guide their tuning. Current research work is also targeting the creation of a self-optimizing compiler that automatically learns the best optimization heuristics based on the behavior of the underlying platform.

Among the tools in the third category is the Active Harmony system. It is a runtime parameter optimization tool that helps focus on the application-dependent parameters that are performance critical. The system tries to improve performance during a single execution based on the observed historical performance data. It can be used to tune parameters such as the size of a read-ahead buffer or what algorithm is being used (e.g., heap sort vs. quick sort). As compared with Active Harmony, the work from Nelson uses a different approach that interacts with the programmer to get high-level models of the impact of parameter values. These models are then used by the system to guide the search for optimization parameters. This approach is called model-guided empirical optimization where models and empirical techniques are used in a hybrid approach.

Popular examples for the last group of autotuning tools are the newly released Parallel Active Harmony, and the Autopilot framework. The Parallel Active Harmony is a combination of the Harmony system and the CHiLL compiler framework. It is an autotuner for scientific codes that applies a search-based autotuning approach. While monitoring the program performance, the system investigates multiple dynamically generated versions of the detected hot loop nests. The performance of these code segments is then evaluated in parallel on the target architecture and the results are processed by a parallel search algorithm. The best candidate is integrated into the application. The second popular example in this group is the Autopilot. It is an integrated toolkit for performance monitoring and dynamical tuning of heterogeneous computational grids based on closed loop control. It uses distributed sensors to extract qualitative and quantitative performance data from the executing applications. This data is processed by distributed actuators and the preliminary performance benchmark is reported to the application developer.

Energy efficiency autotuning. Multi-Petascale supercomputers consist of more than one hundred thousand processing cores and will consume many MW of electrical power. Energy efficiency will be crucial for both cost and environmental reasons, and may soon become as important as pure peak performance. This is exemplified by the fact that since a few years the TOP500 list (<http://www.top500.org/>) also contains power consumption values. Current procurements for high-end supercomputers show that the cost for electricity and cooling is nearly as high as for the hardware, particularly in countries with high energy costs such as Germany. Power consumption is considered one of the greatest challenges on the road to exascale systems.

Dynamic frequency and voltage scaling provides a mechanism to operate modern processors across a broad range of clock frequencies and voltage levels, allowing to trade off performance vs. energy consumption. Overall frequency scaling ideas are based on Advanced Configuration and Power Interface (ACPI, <http://www.acpi.info/>) specification with Intel's SpeedStep implementation or Cool'n'Quiet by AMD, respectively. Processors like Intel's Sandy Bridge are fully compliant with ACPI. Sets of utilities to exploit these techniques are available, and ideas to use them for complete jobs in multi user HPC clusters have already been described.

Whereas dynamic frequency scaling is commonly used in laptops, the impact and usability in HPC is still quite challenging. For applications using several hundreds or thousands of cores, uncoordinated manipulation of the frequency by some background daemon would introduce a new source of OS jitter. Moreover, changing the processor frequency requires on

the order of milliseconds and only yields a benefit if a major part of an application can be run in a given mode continuously. Typically, lowering the CPU frequency can yield a 10% decrease in power consumption while increasing the application runtime by less than 1%. However, the impact of lowering the frequency and voltage on the application performance depends on whether it is CPU, memory, cache or I/O bound. Code regions that are CPU or cache bound can take advantage of higher frequencies, whereas regions that are memory or I/O bound experience only minor performance impacts when reducing the frequency. Therefore it is essential to identify applications and those parts of them that are appropriate for running within a specific power envelope without sacrificing too much performance.

Different metrics for performance, cost, energy, power, cooling and thermal conditions may apply for different usage and optimization scenarios e.g.

- minimizing the energy consumption by reducing the performance of an application by a given percentage
- considering outside temperature conditions, i.e. if it is cold outside and free cooling is applied, an increased power consumption by the compute nodes might be tolerated
- optimizing the total cost of ownership (including baseline investment, power and cooling) for given throughput requirements.

It is quite cumbersome to investigate all these conditions and the various frequency settings manually. Therefore automatic tools are required to automatically identify suitable applications and particular code regions, and finally automatically tune the frequency and power settings to yield optimal results for the desired objectives.

Thematic Sessions

The seminar was organized as a series of thematic sessions. An initial session comprised two overview presentations about performance analysis and measurement tools as well as a general introduction to autotuning, setting the overall context for the seminar. A session on support tools covered code restructuring techniques, testing environments, and performance repositories for autotuning. Two sessions on infrastructures provided insights into frameworks and environments, language support for autotuning as well challenges and requirements in the context of very large-scale systems. A session on energy efficiency tuning gave insight into the challenges and recent developments in optimizing HPC systems and applications with respect to energy consumption. A session on accelerator tuning covered various issues in tuning for GPUs and accelerated parallel systems. A session on techniques covered various topics related to performance-guided tuning, modeling, and scalability. A session on tools covered recent developments in empirical autotuning, semantics support for performance tools and autotuners as well as synthesis of libraries. Various topics related to the tuning of message-passing applications and I/O-related autotuning were covered in a session on MPI and I/O tuning. The session on compiler transformations covered compiler transformations for multi-objective tuning, techniques for tuning irregular applications, as well as on language and compilation support for analysis of semantic graphs.

2 Table of Contents

Executive Summary

<i>Siegfried Benkner, Franz Franchetti, Hans Michael Gerndt, and Jeffrey K. Hollingsworth</i>	214
---	-----

Overview of Talks

Autotuning of Pipeline Patterns for Heterogeneous Manycore Architectures <i>Enes Bajrovic</i>	222
The Autotune Project <i>Siegfried Benkner</i>	222
High-Productivity and High-Performance Analysis of Filtered Semantic Graphs <i>Aydn Buluç</i>	223
HPCToolkit: Performance Tools for Tuning Applications on Heterogeneous Supercomputers <i>Milind Chabbi</i>	224
Blue Gene Performance Data Repository <i>I-hsin Chung</i>	224
Online Automatic Optimization of MPI Runtime Parameters with MPIT <i>Isaias Alberto Compres Urena</i>	225
Towards locality-based autotuning of irregular applications on HPC systems <i>Guojing Cong</i>	225
INSIEME: A compilation and runtime system for multi-objective autotuning for parallel programs <i>Thomas Fahringer</i>	225
Towards Automating Black Belt Programming <i>Franz Franchetti</i>	226
Crowdsourcing autotuning: challenges and possible solutions <i>Grigori Fursin</i>	227
Guiding Tuning with Semi-Analytic Performance Modeling <i>Torsten Hoefler</i>	228
Getting More Auto into Autotuning <i>Jeffrey K. Hollingsworth</i>	229
Performance Modeling for Performance Autotuning <i>Paul D. Hovland</i>	229
Automatic Tuning for GPU BLAS kernels <i>Toshiyuki Imamura</i>	229
ppOpen-AT: Yet Another Directive-base AT Language <i>Takahiro Katagiri</i>	230
Energy-Aware HPC – A Tools Perspective <i>Michael Knobloch</i>	230
Potentials and Limitations for Energy Efficiency Auto-Tuning <i>Andreas Knuepfer</i>	232

Bench-testing Environment for Automated Software Tuning (BEAST) <i>Jakub Kurzak</i>	232
Knowledge-based Performance Engineering and Empirical Autotuning <i>Allen D. Malony</i>	233
Dynamic Tuning for Large-Scale Computing using ELASTIC <i>Andrea Martinez</i>	233
Opportunities and Strategies for I/O Auto-Tuning <i>Renato Miceli Costa Ribeiro</i>	234
Parallel Performance Measurement and Analysis on Extreme-Scale Computer Systems <i>Bernd Mohr</i>	234
Restructuring Legacy Codes to Enable Autotuning <i>Shirley V. Moore</i>	235
Autotuning the Energy Consumption <i>Carmen Navarrete</i>	235
High performance program generators: systematic construction with language support <i>Georg Ofenbeck</i>	236
Methodology for MPI Applications Autotuning <i>Antonio Pimenta</i>	237
Exploiting processor inhomogeneity for performance optimization under a power bound <i>Barry Roundtree</i>	237
Tuning support for a Xeon Phi template library <i>Martin Sandrieser</i>	237
Performance Analysis, Energy Efficiency Optimizations and Auto-Tuning <i>Robert Schoene</i>	238
Providing the Necessary Semantic Context for Performance Tools and Autotuners <i>Martin Schulz</i>	238
Smarter Libraries with Synthesis <i>Armando Solar-Lezama</i>	238
Application-independent Autotuning for GPUs <i>Walter F. Tichy</i>	239
Generalized roofline analysis? <i>Richard Vuduc</i>	239
Autotuning for Scale <i>Felix Wolf</i>	239
Working Groups	
Energy and power autotuning	240
Language support for Autotuning	241
Infrastructures	241

Siegfried Benkner, Franz Franchetti, H. Michael Gerndt, and Jeffrey K. Hollingsworth 221

Black/Whitebox Autotuning 242
Search Algorithms 242
Participants 244

3 Overview of Talks

3.1 Autotuning of Pipeline Patterns for Heterogeneous Manycore Architectures

Enes Bajrovic (Universität Wien, AT)

License  Creative Commons BY 3.0 Unported license
© Enes Bajrovic

We have developed a high-level, component-based approach to programming heterogeneous manycore architectures with support for parallel patterns. Central to this approach are multi-architectural components, which encapsulate different implementation variants of application functionality tailored by expert programmers for different core types of a heterogeneous system. End users construct applications at a high level of abstraction using component interfaces and high-level coordination primitives, which are then mapped to a task-based runtime that selects and dynamically schedules the best component implementation variants for efficient parallel execution on a heterogeneous manycore architecture. In this talk we present our ongoing work on automatic performance tuning of pipeline patterns and the underlying pattern runtime for CPU/GPU based architectures as currently done within the European AutoTune project.

References

- 1 R. Miceli, G. Civario, A. Sikora, E. Cesar, M. Gerndt, H. Haitof, C. Navarrete, S. Benkner, M. Sandrieser, L. Morin, and F. Bodin. *AutoTune: A Plugin-Driven Approach to the Automatic Tuning of Parallel Applications*, In *Applied Parallel and Scientific Computing* (P. Manninen and P. Oester, eds.), vol. 7782 of *LNCIS*, pp. 328–342, Springer, 2013.
- 2 S. Benedict, V. Petkov, and M. Gerndt. *PERISCOPE: An Online-Based Distributed Performance Analysis Tool*. In: *Tools for High Performance Computing 2009* (M. S. Mueller, M. M. Resch, A. Schulz, and W. E. Nagel, eds.), pp. 1–16, Springer, 2010
- 3 E. Bajrovic, S. Benkner, J. Dokulil, M. Sandrieser. *Autotuning of Pattern Runtimes for Accelerated Parallel Systems*. In: *PARCO 2013*, September 2013, Munich, Germany, 10-13 Sept 2013, Munich, Germany (2013)
- 4 S. Benkner, E. Bajrovic, E. Marth, M. Sandrieser, R. Namyst, and S. Thibault. *High-Level Support for Pipeline Parallelism on Manycore Architectures*. In: *Euro-Par 2012*, vol. 7484, pp. 614–625, 2012.
- 5 S. Benkner, S. Pllana, J. L. Traeff, P. Tsigas, U. Dolinsky, C. Augonnet, B. Bachmayer, C. Kessler, D. Moloney, V. Osipov. *PEPPHER: Efficient and Productive Usage of Hybrid Computing Systems*. *IEEE Micro*, vol. 31, no. 5, pp. 28–41, Sep./Oct. 2011, DOI: 10.1109/MM.2011.67.

3.2 The Autotune Project

Siegfried Benkner (Universität Wien, AT)

License  Creative Commons BY 3.0 Unported license
© Siegfried Benkner

The European AutoTune project develops the Periscope Tuning Framework (PTF) for performance and energy efficiency tuning of parallel applications on current and future multicore-based architectures. PTF extends Periscope, an automatic online and distributed

performance analysis tool, with tuning plugins for automatic performance and energy efficiency tuning, closing the gap between performance analysis and tuning. The tuning plugins developed within the project include a GPU tuning plugin for HMPP/OpenCL codes, a plugin for tuning of pipeline patterns for heterogeneous manycore architectures, an energy efficiency tuning plugin, an MPI tuning plugin, and a compiler options exploration plugin. Wherever possible, the whole tuning process, consisting of automatic performance analysis and automatic tuning, will be executed online, i.e., during a single run of the application. This talk will present an overview of PTF, the underlying tuning model, and selected tuning plugins.

References

- 1 R. Miceli, G. Civario, A. Sikora, E. Cesar, M. Gerndt, H. Haitof, C. Navarrete, S. Benkner, M. Sandrieser, L. Morin, and F. Bodin. *AutoTune: A Plugin-Driven Approach to the Automatic Tuning of Parallel Applications*, In *Applied Parallel and Scientific Computing* (P. Manninen and P. Oester, eds.), vol. 7782 of *LNC3*, pp. 328–342, Springer, 2013.
- 2 S. Benedict, V. Petkov, and M. Gerndt. *PERISCOPE: An Online-Based Distributed Performance Analysis Tool*. In: *Tools for High Performance Computing 2009* (M. S. Mueller, M. M. Resch, A. Schulz, and W. E. Nagel, eds.), pp. 1–16, Springer, 2010
- 3 E. Bajrovic, S. Benkner, J. Dokulil, M. Sandrieser. *Autotuning of Pattern Runtimes for Accelerated Parallel Systems*. In: *PARCO 2013*, September 2013, Munich, Germany, 10-13 Sept 2013, Munich, Germany (2013)
- 4 S. Benkner, E. Bajrovic, E. Marth, M. Sandrieser, R. Namyst, and S. Thibault. *High-Level Support for Pipeline Parallelism on Manycore Architectures*. In: *Euro-Par 2012*, vol. 7484, pp. 614–625, 2012.

3.3 High-Productivity and High-Performance Analysis of Filtered Semantic Graphs

Aydın Buluç (Lawrence Berkeley National Laboratory, US)

License © Creative Commons BY 3.0 Unported license
© Aydın Buluç

Joint work of Buluç, Aydın; Duriakova, Erika; Fox, Armando; Gilbert, John; Kamil, Shoaib; Lugowski, Adam; Oliker, Leonid; Williams, Samuel

Main reference A. Buluç, E. Duriakova, A. Fox, J. R. Gilbert, S. Kamil, A. Lugowski, L. Oliker, S. Williams, “High-Productivity and High-Performance Analysis of Filtered Semantic Graphs,” in *Proc. of the IEEE 27th Int’l Parallel and Distributed Processing Symp. (IPDPS’13)*, pp. 237–248, IEEE CS, 2013.

URL <http://dx.doi.org/10.1109/IPDPS.2013.52>

Graph theory is used to model large-scale complex systems in various scientific domains. Filtering on attributed semantic graphs enable a broad set of real applications that maintain important auxiliary information through attributes on individual edges and vertices. We achieve high performance in a high- productivity Python-based graph library (KDT) by utilizing a highly optimized sparse linear algebra based backend (Combinatorial BLAS), and automatically translating analytic queries via selective just-in-time embedded specialization (SEJITS).

References

- 1 Aydın Buluç, Erika Duriakova, Armando Fox, John Gilbert, Shoaib Kamil, Adam Lugowski, Leonid Oliker, and Samuel Williams. High-productivity and high-performance analysis of filtered semantic graphs. In *Proceedings of the IPDPS*. IEEE Computer Society, 2013.

3.4 HPCToolkit: Performance Tools for Tuning Applications on Heterogeneous Supercomputers

Milind Chabbi (Rice University, US)

License © Creative Commons BY 3.0 Unported license
© Milind Chabbi

Joint work of Chabbi, Milind; Murthy, Karthik; Fagan, Michael; Mellor-Crummey, John
Main reference M. Chabbi, K. Murthy, M. Fagan, J. Mellor-Crummey, “Effective Sampling-Driven Performance Tools for GPU-Accelerated Supercomputers,” in Proc. of Int’l Conf. for High Performance Computing, Networking, Storage and Analysis (SC’13), pp. 43:1–43:12, ACM, 2013.
URL <http://dx.doi.org/10.1145/2503210.2503299>

Achieving higher performance under limited power is an open challenge for exascale computing. Modern supercomputers employ accelerators towards achieving this goal. Tuning applications employing CPUs and GPUs across an array of nodes demands a systemic performance perspective. In this talk, I introduce three key ideas developed by Rice University’s HPCToolkit team to address these issues. First, we introduce CPU-GPU blame shifting – a technique to pinpoint and quantify idleness arising due to non overlapped CPU and GPU computations within a node. Second, we introduce a technique to identify and assess the impact of small set of sporadically poorly behaving nodes sabotaging the performance of entire application. Finally, we introduce a technique to model the performance impact of multiplexing GPUs among multiple processes.

3.5 Blue Gene Performance Data Repository

I-hsin Chung (IBM TJ Watson Research Center – Yorktown Heights, US)

License © Creative Commons BY 3.0 Unported license
© I-hsin Chung

As the high performance computer architectures become more complicated and larger in scale, it is a challenge to understand the utilization of system resources by applications. In this presentation, we introduce the Blue Gene Performance Data Repository, which automatically collects a wide range of performance information and stores it into a relational database for subsequent analysis. With minimal overhead to the users, the database collects performance data from different aspects of application execution including MPI, OpenMP, hardware counter, etc. With a standardized and uniform storage format, the database supports a wide range of queries and presentations. We will show examples of how this database can help users understand the application performance behavior and the system hardware usage. This information can also be used for software/hardware co-design for next generation systems. The Blue Gene Performance Data Repository is available to Blue Gene/Q users under an open source license.

3.6 Online Automatic Optimization of MPI Runtime Parameters with MPIT

Isaias Alberto Compres Urena (TU München, DE)

License © Creative Commons BY 3.0 Unported license
© Isaias Alberto Compres Urena

The point-to-point performance of MPI applications can be improved by adjusting the internal communication parameters of the MPI runtime. Point-to-point thresholds are set on good known values; however, the optimum value for a specific system may differ from the defaults. Performance can be further improved by selecting alternative internal implementations of collective operations. Default internal collective operations are set for specific process counts and buffer sizes and are also not always optimal. The space generated by all combinations of thresholds and internal algorithms is large; however, the search for optima for a particular application can be reduced based on its use of the MPI. The optimization process can be further accelerated by exposing the relevant parameters through MPIT and performing the search online. Empirical data, collected on the SuperMUC, show that significant improvements in MPI performance can be achieved versus the default settings. These results were gathered with a modified version of the latest release of MPICH2.

3.7 Towards locality-based autotuning of irregular applications on HPC systems

Guojing Cong (IBM TJ Watson Research Center – Yorktown Heights, US)

License © Creative Commons BY 3.0 Unported license
© Guojing Cong

As more analytics workloads emerge, tuning irregular applications for high performance becomes increasingly critical for the utilization of HPC systems. In this talk we show that locality-based tuning is a unified approach that can reduce communication overhead and improve cache performance simultaneously. We use multithreaded applications on cache-based NUMA architectures as our examples.

3.8 INSIEME: A compilation and runtime system for multi-objective autotuning for parallel programs

Thomas Fahringer (Universität Innsbruck, AT)

License © Creative Commons BY 3.0 Unported license
© Thomas Fahringer

Joint work of Jordan, Herbert; Thoman, Peter; Durillo, Juan; Gschwandtner, Philipp; Pellegrini, Simone; Fahringer, Thomas; Moritsch, Hans

Main reference H. Jordan, P. Thoman, J. J. Durillo, S. Pellegrini, P. Gschwandtner, T. Fahringer, H. Moritsch, “A multi-objective autotuning framework for parallel codes,” in Proc. of the Int’l Conf. on High Performance Computing, Networking, Storage and Analysis (SC’12), pp. 10:1–10:12, IEEE CS, 2012.

URL <http://dl.acm.org/citation.cfm?id=2389010>

URL <http://www.insieme-compiler.org>

Efficient parallelization and optimization for modern parallel architectures is a time-consuming and error-prone task that requires numerous iterations of code transformations, tuning and

performance analysis which in many cases have to be redone for every different target architecture.

We introduced an innovative autotuning compiler named INSIEME which consists of a compiler component featuring a multi-objective optimizer and a runtime system. It consists of a compiler component featuring a multi-objective optimizer and a runtime system. The multi-objective optimizer derives a set of non-dominated solutions, each of them expressing a trade-off among the different conflicting objectives such as execution time, energy consumption and efficiency. This set is commonly known as Pareto set in the field of multi-objective optimization research. Our search algorithm, which explores code transformations and their parameter settings, dynamic concurrency throttling (DCCT), and dynamic voltage and frequency scaling (DVFS) is based on Differential Evolution. Additionally, Rough sets are employed to reduce the search space, and thus the number of evaluations required during compilation. To make effective use of the resulting Pareto set of optimal solutions, each of them has to be made available at runtime. This is achieved by having the compiler generate a set of code versions per region, each corresponding to one specific solution. The runtime system then exploits the trade-off among the different objectives by selecting a specific solution (code version) for each region, based on context-specific criteria. We have implemented our techniques based on the Insieme Compiler and Runtime infrastructure. Our approach is generic and can be applied to arbitrary transformations and parameter settings. We demonstrate our approach by tuning loop tiling, the nr. of threads, and clock frequency in cache sensitive parallel programs, optimizing for runtime, efficiency and energy.

The major contributions of this work include:

- the design of a novel autotuning architecture facilitating the consideration of multiple conflicting criteria simultaneously by interpreting the central task as a multi-objective optimization problem
- the combination of the search for optimal tile sizes, clock frequency settings, and the ideal number of threads for a parallel code section to minimize execution time and energy consumption, and maximize parallel efficiency into a single, multi-objective optimization problem
- the development of a multi-objective optimization algorithm capable of solving the combined problem using a reasonable number of iterative compilation steps

3.9 Towards Automating Black Belt Programming

Franz Franchetti (Carnegie Mellon University, US)

License © Creative Commons BY 3.0 Unported license

© Franz Franchetti

Main reference W. Yu, F. Franchetti, J. C. Hoe, T. Chen, “Highly Efficient Performance Portable Tracking of Evolving Surfaces,” in Proc. of the 26th Int’l Parallel and Distributed Processing Symp. (IPDPS’12), pp. 296–307, IEEE CS, 2012.

URL <http://dx.doi.org/10.1109/IPDPS.2012.36>

Only a select few performance programmers in major processor vendors’ software divisions, universities, national laboratories, and specialized companies have the skill set to achieve high efficiency on today’s processors. Due to the labor-intensive nature of the work, the fast rate of newly arriving platforms, and the fact that performance tuning is more a black art than science, only the most important fundamental computation functions can be fully optimized. Today we are farther away than ever from John Backus’ design criterion for the first Fortran compiler to automatically achieve close-to-human performance. Automatic

performance tuning has bridged this gap for a few well-understood computational kernels like matrix multiplication, fast Fourier transform, and sparse matrix-vector multiplication, where systems like ATLAS, SPIRAL, FFTW, and OSKI showed that it is possible for automatic systems to compete with code that has been hand-tuned by “black belt programmers”, which extracted the full performance potential from the target machines.

In this talk we investigate how black belt performance levels can be obtained automatically for irregular kernels with highly data-dependent control flow that operate on dynamic data structures, which makes the usual optimization methods impossible to apply. We focus on three illustrative examples: The first example is evaluating a logical equation in conjugate normal form, which is expressed as reduction across nested linked lists. The second example is the evolution of an interface surface inside a volume (e.g., a shock wave of an explosion), which translates into stencil operations on a contiguous sparse subset of pixels of a dense regular grid. The third example is a Monte Carlo simulation-based probabilistic power flow computation for distribution networks. In all cases we achieve a high fraction of machine peak, at the cost of applying aggressive optimization techniques that so far are beyond the capabilities of automatic tools. Autotuning and program generation played a major role in obtaining the final optimized implementation, as the necessary optimization techniques have a large parameter space and require extensive code specialization. In all cases the original code consisted of a few lines of C code, while the final code comprises hundreds to thousands of lines of architecture-specific SIMD intrinsic code and OpenMP or PThreads parallelization with custom synchronization. We extract lessons from the three examples on how to design future autotuning and program generation systems that will automate the optimization of such kernels.

References

- 1 W. Yu, F. Franchetti, J. C. Hoe, T. Chen. *Highly Efficient Performance Portable Tracking of Evolving Surfaces*. Proceedings of the 26th International Parallel and Distributed Processing Symposium (IPDPS), 296-307, 2012.
- 2 T. Cui and F. Franchetti. *Autotuning a Random Walk Boolean Satisfiability Solver*. Proceedings of The Sixth International Workshop on Automatic Performance Tuning (iWAPT), 2011.

3.10 Crowdsourcing autotuning: challenges and possible solutions

Grigori Fursin (INRIA Saclay – Île-de-France – Orsay, FR)

License © Creative Commons BY 3.0 Unported license

© Grigori Fursin

Main reference G. Fursin, “Collective Mind: cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning,” INRIA technical report, arXiv:1308.2410v1 [cs.SE], 2013; also available on HAL (HAL-00850880).

URL <http://arxiv.org/abs/1308.2410>

URL <http://hal.inria.fr/hal-00850880>

Empirical program and architecture tuning combined with run-time adaptation and machine learning has been demonstrating good potential to improve performance, power consumption and other important metrics of computer systems for more than a decade. However, it is still far from the widespread production use due to unbearably long exploration and training times, dramatic growth in the amount of experimental data (“big data”), ever changing tools and their interfaces, lack of a common experimental methodology, and lack of unified mechanisms

for knowledge building and exchange apart from publications where reproducibility of results is often not even considered.

We present our long-term holistic vision and a plugin-based Collective Mind infrastructure with unified web services to systematize characterization and optimization of computer systems through crowd tuning, extensible repositories of knowledge, and machine learning. In this cooperative approach, multi-objective program and architecture characterization and tuning is transparently distributed among many participants while consolidating and unifying existing techniques and tools, and utilizing any available mobile, cluster or cloud computer services for online learning and classification. Any unexpected behavior is analyzed using shared data mining and predictive modeling plugins or exposed to the community at cTuning.org for collaborative explanation. Gradually increasing optimization knowledge helps to continuously improve optimization heuristics of any compiler, predict optimizations for new programs or suggest efficient run-time adaptation strategies depending on end-user requirements. It also allows researchers to quickly reproduce and validate existing results, and focus their effort on novel approaches combined with data mining, classification and predictive modeling. At the same time, it allows conferences and journals to favor publications that can be collaboratively validated by the community.

We initially validated this concept in several academic projects including EU FP6 MILE-POST to build first publicly available machine learning based self-tuning compiler, and later in industry in collaboration with IBM, ARC (Synopsys), CAPS Enterprise, Intel/CEA Exascale Lab, STMicroelectronics and ARM. Since 2007, we started sharing all our past research artifacts including hundreds of codelets, numerical applications, data sets, models, universal experimental pipelines for adaptive optimization space exploration and autotuning, self-tuning machine learning based meta compiler, and unified statistical analysis, classification and predictive modeling plugins in a public Collective Mind repository (c-mind.org/repo). Therefore, we also present and discuss various encountered problems, pitfalls and possible long-term solutions when crowdsourcing autotuning.

3.11 Guiding Tuning with Semi-Analytic Performance Modeling

Torsten Hoefler (ETH Zürich, CH)

License  Creative Commons BY 3.0 Unported license
© Torsten Hoefler

Any form of automated tuning often faces a parameter space that is too large to be searched completely. We propose to utilize manual and automated semi-analytic performance modeling techniques to guide the tuner through the search-space. We use those methods to guide an approximate search to limit the space that then has to be considered for a full search. Our techniques can be used for manual as well as automated tuning approaches. We remark that manual tuning is often more powerful since it can operate at the algorithm level. We show several approaches to the problem and how to integrate the modeling into the development and tuning workflow. Our techniques may lead to much more efficient and quickly converging automated tuning.

3.12 Getting More Auto into Autotuning

Jeffrey K. Hollingsworth (University of Maryland – College Park, US)

License © Creative Commons BY 3.0 Unported license
© Jeffrey K. Hollingsworth

Joint work of Hollingsworth, Jeffrey K.; Chaen, Ray S.

Main reference J. K. Hollingsworth, “Towards Fully Automatic Auto-tuning: Leveraging language features of Chapel Ray Chen,” *Int’l Journal of High Performance Computing Applications*, 27(4):394–402, Nov. 2013.

URL <http://dx.doi.org/10.1177/1094342013493198>

Nearly twenty five years ago I started working in performance tuning to make it easier to get programs to run well. Still tuning was tedious, so more than a decade ago I starting working on autotuning to make it easier to get programs to run well. Autotuning made it possible to mechanize the exploration of a space of tuning options. However, the process of identifying those options is still too manual and tedious. In this talk, I will argue that the next big issue in autotuning is making it truly automatic.

3.13 Performance Modeling for Performance Autotuning

Paul D. Hovland (Argonne National Laboratory, US)

License © Creative Commons BY 3.0 Unported license
© Paul D. Hovland

We provide some thoughts on performance modeling in the context of automatic performance tuning. We consider analytic models constructed through source code analysis, semi-analytic models constructed with a combination of source code analysis and empirical measurement, and fully empirical models. We imagine several uses for performance models in conjunction with autotuning, including surrogate construction for algorithm evaluation, surrogate-based search, and bounds analysis for search space truncation.

3.14 Automatic Tuning for GPU BLAS kernels

Toshiyuki Imamura (RIKEN – Kobe, JP)

License © Creative Commons BY 3.0 Unported license
© Toshiyuki Imamura

Development of a GPGPU numerical kernel is a typical example of performance autotuning. In this talk, the author will present the development of CUDA BLAS kernels, especially Level 2 kernels, which are bounded by memory bandwidth and have numerous parameter space to be searched. Sieving the number of parameter set and selecting appropriate faster kernel code is significant. Based on a two-stage sieving scheme, we eventually obtained the best-tuned kenrnels, which outperform the state-of-the-art CUDABLAS libraries like CUBLAS and MAGMABLAS. We conclude that better optimization has been carried out by automatic tuning.

3.15 ppOpen-AT: Yet Another Directive-base AT Language

Takahiro Katagiri (University of Tokyo, JP)

License © Creative Commons BY 3.0 Unported license
© Takahiro Katagiri

Main reference T. Katagiri, S. Ito, S. Ohshima, “Early Experiences for Adaptation of Auto-tuning by ppOpen-AT to an Explicit Method,” in Proc. of the IEEE 7th Int’l Symp. on Embedded Multicore SoCs (MCSoc’13), pp. 153–158, IEEE CS, 2013.

URL <http://dx.doi.org/10.1109/MCSoc.2013.15>

URL <http://ppopenhpc.cc.u-tokyo.ac.jp/>

Although several Autotuning (AT) languages have been proposed and studied for the last decade, we do not still have crucial languages and tools for AT for supercomputers in operation. In this presentation, we present another AT language based on directive-base language, which is ppOpen-AT[1]. ppOpen-AT is designed with based on basic functions of ABCLibScript[2]. Several new AT functions are developed with taking into account kernels of application software. With respect to supercomputer environments, code generator and AT framework of ppOpen-AT do not require any daemons and script languages for code generation and AT cycle phases. The new developed AT function of ppOpen-AT is loop transformation, such as loop split and loop fusion. This is not new function for loop transformation. However, the loop split function we focus on in ppOpen-AT breaks data flow-dependencies in original loop. In particular, the split requires increase of computations when the loop is split in some examples. Hence, vendor compilers cannot supply the function to perform the loop split. To establish the AT with loop split function, we use developers knowledge via directives of ppOpen-AT. As a scenario of AT, we adapt the AT function to a real application based on finite difference method (FDM). By using the Fujitsu PRIMEHPC FX10, that is supercomputer in operation in Information Technology Center, the University of Tokyo, we obtain essential performance improvement by adapting the AT function. We also present preliminary results for the Fujitsu FX10, the Sandy Bridge, and the Xeon Phi.

References

- 1 T. Katagiri, S. Ito, S. Ohshima. *Early Experiences for Adaptation of Autotuning by ppOpen-AT to an Explicit Method*. Special Session: Auto-Tuning for Multicore and GPU (ATMG) (In Conjunction with the IEEE MCSoc-13), Proceedings of MCSoc-13, 2013.
- 2 T. Katagiri, K. Kise, H. Honda, T. Yuba. *ABCLibScript: A Directive to Support Specification of an Autotuning Facility for Numerical Software*. Parallel Computing, Vol. 32, Issue 1, 92-112, 2006.

3.16 Energy-Aware HPC – A Tools Perspective

Michael Knobloch (Jülich Supercomputing Centre, DE)

License © Creative Commons BY 3.0 Unported license
© Michael Knobloch

The power consumption and energy-efficiency of supercomputers have been a major topic in High-Performance Computing (HPC) in recent years [1], with the target of 20 MW for an Exascale system. This goal still requires power and energy-efficiency improvements of two orders of magnitude and can only be reached by hardware and software working closely together. Additionally, tuning for power and tuning for energy consumption might require different tools and analyzes.

In this talk I present the past, present and future of the work done at the Jülich Supercomputing Centre (JSC), one of the largest supercomputing centres in Europe, in terms of tools to measure and optimize power and energy consumption of HPC applications. JSC is involved in energy-efficiency related projects since 2009, when awareness for energy-aware HPC started in Germany and the first projects were funded and continues to research ways to reduce the energy-efficiency of its systems.

The eeClust (Energy-Efficient Cluster Computing) project [2, 3], the first project I present, had the goal to analyze resource usage of HPC applications and turn unused components to lower power states when not used [4]. An integral part of this work was extending the existing performance analysis tool-sets Vampir [5] and Scalasca [6] to collect and display power and energy consumption data and determine energy-saving potential within applications [7].

The second project I present is the Exascale Innovation Centre (EIC), a joint collaboration of JSC and IBM research Germany with the goal of co-design the next generation of supercomputers, which has one work package dealing with energy efficiency. Here we investigated the power consumption of IBM machines like Blue Gene/P [8] and POWER7 [9]. We further used these results to generate a model for the CPU power consumption of the POWER7 [10].

Third, I give an outlook to Score-E, a project which will start in October 2013. One major goal of this project is to extend Score-P [11], the new community-driven measurement system which is used by Vampir and Scalasca, to capture and store events related to power and energy consumption. Further, it's planned to enhance the power and energy consumption modelling and prediction as well as to develop new ways of visualization of performance and power data.

I finally conclude with an overview of the lessons we learned during these years and outline the requirements tools pose to future hardware generations in order to be most useful and productive.

References

- 1 Knobloch, M. *Energy-Aware High Performance Computing – A Survey*. Green and Sustainable Computing: Part II, Academic Press, 2013
- 2 Minartz, T. and Molka, D. and Knobloch, M. and Krempel, S. and Ludwig, T. and Nagel, W.E. and Mohr, B. and Falter, H. *eeClust: Energy-Efficient Cluster Computing*. Competence in High Performance Computing 2010, (CiHPC-2010), Proceedings of an International Conference on Competence in High Performance Computing, June 2010
- 3 Knobloch, M. and Minartz, T. and Molka, D. and Krempel, S. and Ludwig, T. and Mohr, B. *eeClust – Energy-efficient Cluster Computing*. Proceedings of the SC11, 2011
- 4 Minartz, T. and Ludwig, T. and Knobloch, M. and Mohr, B. *Managing hardware power saving modes for high performance computing*. Green Computing Conference and Workshops (IGCC), 2011
- 5 Knüpfer, A. and Brunst, H. and Doleschal, J. and Jurenz, M. and Lieber, M. and Mickler, H. and Müller, M. and Nagel, W.E. *The Vampir Performance Analysis Tool-Set*. Tools for High Performance Computing, Springer, 2008
- 6 Geimer, M. and Wolf, F. and Wylie, B.J.N. and Abraham, E. and Becker, D. and Mohr, B. *The Scalasca performance toolset architecture*. Concurrency and Computation: Practice and Experience, Wiley, 2011
- 7 Knobloch, M. and Mohr, B. and Minartz, T. *Determine energy-saving potential in wait-states of large-scale parallel programs*. Computer science – research and development, Springer, 2012

- 8 Hennecke, M. and Frings, W. and Homberg, W. and Zitz, A. and Knobloch, M. and Böttiger, H. *Measuring power consumption on IBM Blue Gene/P*. Computer science – research and development, Springer, 2012
- 9 Knobloch, M. and Foszczynski, M. and Homberg, W. and Pleiter, D. and Böttiger, H. *Mapping fine-grained power measurements to HPC application runtime characteristics on IBM POWER7*. Computer science – research and development, Springer, 2013
- 10 Gschwandtner, P. and Knobloch, M. and Mohr, B. and Pleiter, D. and Fahringer, T. *Modeling CPU Energy Consumption of HPC Applications on the IBM POWER7*. to appear
- 11 Knüpfer, A. and Rössel, C. and an Mey, D. and Biersdorff, S. and Diethelm, K. and Eschweiler, D. and Geimer, M. and Gerndt, M. and Lorenz, D. and Malony, A.D. and Nagel, W.E. and Oleynik, Y. and Philippen, P. and Saviankou, P. and Schmidl, D. and Shende, S.S. and Tschüter, R. and Wagner, M. and Wesarg, B. and Wolf, F. *Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir*. Proc. of 5th Parallel Tools Workshop, Springer, 2012

3.17 Potentials and Limitations for Energy Efficiency Auto-Tuning

Andreas Knuepfer (TU Dresden, DE)

License  Creative Commons BY 3.0 Unported license
© Andreas Knuepfer

With an objective target of 20 MW for an Exascale system, the power budget for future super computers has a severe restriction. Fortunately, current hardware and runtime environments feature various adjustable parameters that can be changed at runtime in order to confine the power usage. We present an overview of the potentials and limitations of several power saving mechanisms and describe the integration of energy efficiency optimizations in existing performance analysis tools. We also introduce an energy efficiency tuning cycle and exemplify its usage.

3.18 Bench-testing Environment for Automated Software Tuning (BEAST)


Jakub Kurzak (University of Tennessee, US)

License  Creative Commons BY 3.0 Unported license
© Jakub Kurzak

The goal of BEAST is to create a framework for exploring and optimizing the performance of computational kernels on hybrid processors that 1) applies to a diverse range of computational kernels, 2) (semi)automatically generates better performing implementations on various hybrid processor architectures, and 3) increases developer insight into why given kernel/processor combinations have the performance profiles they do. We call this form of optimization “bench-tuning” because it builds on the model used for traditional benchmarking by combining an abstract kernel specification and corresponding verification test with automated testing and data analysis tools to achieve this threefold goal.

3.19 Knowledge-based Performance Engineering and Empirical Autotuning

Allen D. Malony (University of Oregon, US)

License  Creative Commons BY 3.0 Unported license
© Allen D. Malony

In the last 20 years, the parallel performance community has delivered powerful techniques and tools for measurement and analysis of parallel systems and applications. While these research and development advances have been important for observing how the evolving features of parallel machines affect application performance, there has been less done by the parallel computing community as a whole to carry forward performance knowledge from technology generation to generation.

Parallel computing is now at a point where it is imperative to incorporate knowledge at all levels in order to engineer optimized applications targeted to the essential features of the environment in which they will run. This talk will discuss the role of empirical autotuning as a process for systematic experimentation to create a knowledge base of performance information that can be applied in support of automatic performance engineering and tuning objectives. Results will be presented on the development of an empirical autotuning framework combining a parallel performance analysis system with a code transformation tool and autotuner environment.

3.20 Dynamic Tuning for Large-Scale Computing using ELASTIC

Andrea Martínez (Autonomous University of Barcelona, ES)

License  Creative Commons BY 3.0 Unported license
© Andrea Martínez

Dynamic tuning is the most viable tactic to improve the performance of parallel applications which present long runnings times or behavioural patterns that change depending on the input data set or according to data evolution. In order to bring this strategy to large scale computers, we propose a model that enables scalable dynamic tuning. This model is based on the application decomposition combined with an abstraction mechanism to solve local and global problems. The proposed model has been implemented in the form of ELASTIC, a tool for large- scale dynamic tuning. Using ELASTIC, an experimental evaluation has been performed over a synthetic parallel application (up to 16384 tasks) and an agent- based real parallel application (up to 2048 tasks). The results demonstrate that the proposed model, embodied in ELASTIC, is able to manage the increasing number of processes, allowing for the effective tuning of large-scale parallel applications.

References

- 1 Andrea Martínez and Anna Sikora and Eduardo César and Joan Sorribes. *How to Scale Dynamic Tuning to Large-Scale Applications*. Proceedings of International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS) – IPDPS, 2013.
- 2 Andrea Martínez and Anna Sikora and Eduardo César and Joan Sorribes. *How to Determine the Topology of Hierarchical Tuning Networks for Dynamic Auto- Tuning in Large-Scale Systems*. Proceedings of International Workshop on Automatic Performance Tuning (IWAPT) – ICCS, 2013.

3.21 Opportunities and Strategies for I/O Auto-Tuning

Renato Miceli Costa Ribeiro (ICHEC – Galway, IR)

License  Creative Commons BY 3.0 Unported license
© Renato Miceli Costa Ribeiro

In the HPC community, I/O issues are both one of the most common bottleneck and scalability limiting factors in codes, and one scientists and developers alike are the least aware of. For this, automating the I/O tuning is likely to lead to significant performance improvements, especially since the corresponding manual tuning is a complex task often out of reach of users and code developers. In this talk we will discuss the opportunities and possible approaches to automatically tuning the I/O of HPC software. We will introduce some of the most common I/O issues one can encounter while developing or using a HPC code, examine their relationship to the machine hardware, OS and filesystem settings, and explore prospective automatic tuning strategies adapted to address each one of these issues.

3.22 Parallel Performance Measurement and Analysis on Extreme-Scale Computer Systems

Bernd Mohr (Jülich Supercomputing Centre, DE)

License  Creative Commons BY 3.0 Unported license
© Bernd Mohr

The number of processor cores available in high-performance computing systems is steadily increasing. In the June 2013 list of the TOP500 supercomputers, only three systems have less than 4,096 processor cores and the average is almost 39,000 cores, which is an increase of 9,000 in just one year. Even the median system size is already over 16,000 cores. While these machines promise ever more compute power and memory capacity to tackle today's complex simulation problems, they force application developers to greatly enhance the scalability of their codes to be able to exploit it. To better support them in their porting and tuning process, many parallel tools research groups have already started to work on scaling their methods, techniques and tools to extreme processor counts. In this talk, we survey existing performance analysis and optimization tools covering both profiling and tracing techniques, report on our experience in using them in extreme scaling environments, review existing working and promising new methods and techniques, and discuss strategies for addressing unsolved issues and problems. Important performance tool sets covered include TAU [1], HPCToolkit [2], Paraver [3], Vampir [4], Scalasca [5], Open|SpeedShop [6], and Periscope [7].

References

- 1 TAU, University of Oregon, US, <http://tau.uoregon.edu>.
- 2 HPCToolkit, Rice University, US, <http://hpctoolkit.org>.
- 3 Extrae/Paraver, BSC, Spain, <http://www.bsc.es/paraver>.
- 4 Vampir, TU Dresden, Germany, <http://www.vampir.eu>.
- 5 Scalasca, JSC, Germany, <http://www.scalasca.org>.
- 6 Open|SpeedShop, Krell Institute, US, <http://www.openspeedshop.org>.
- 7 Periscope, TU Munich, Germany, <http://www.lrr.in.tum.de/~periscop/>.

3.23 Restructuring Legacy Codes to Enable Autotuning

Shirley V. Moore (University of Texas – El Paso, US)

License © Creative Commons BY 3.0 Unported license
© Shirley V. Moore

Although autotuning has been applied successfully in the domain of linear algebra software, its use with general application codes, especially parallel codes, has had limited success. In many cases, the parallel implementation obscures the inherent parallelism and introduces false data dependencies and unnecessary synchronization that can be hard to detect and eliminate. In this talk, we discuss ways of restructuring legacy codes to increase opportunities for autotuning and give examples from the domain of computational chemistry.

3.24 Autotuning the Energy Consumption

Carmen Navarrete (Leibniz Rechenzentrum – München, DE)

License © Creative Commons BY 3.0 Unported license
© Carmen Navarrete
Joint work of Navarrete, Carmen; Guillen, Carla; Hesse, Wolfram; Brehm, Matthias

Saving energy in high performance systems is becoming increasingly important as systems become larger and energy costs rise. Although modern processors with Dynamic Voltage and Frequency Scaling (DVFS) already automatically scale the frequency depending on the load of the processor, they do not detect the effects of this scaling in a parallel application as a whole. Manual efforts to tune the processor frequency of a parallel application to obtain the best energy to solution can be extremely time consuming. Thus it is important to have analysis tools which help measure, analyse and propose solutions to the user. The Periscope Tuning Framework, henceforth called PTF, has the capabilities to automatically search for optimizations in a code. This tool can be used to define a structured analysis mechanism which will automatically search for an optimal solution. This mechanism can be coded as a plugin and can be easily integrated into the PTF. We present a plugin for the PTF which will enable the detailed analysis of a parallel code, i.e. per code region and task. The tool will automatically propose processor frequencies at the level of regions and tasks which optimize the energy to solution of a parallel application as a whole.

References

- 1 Miceli et al. *AutoTune: A Plugin-Driven Approach to the Automatic Tuning of Parallel Applications*. In Applied Parallel and Scientific Computing, Vol. 7782 (2013), pp. 328–342, Springer Berlin Heidelberg, 2013.
- 2 Tiwari et al. *Auto-tuning for energy usage in scientific applications*. Proceedings of the 2011 international conference on Parallel Processing – Euro-Par’11, Volume 2, pp. 178–187, Springer-Verlag, Berlin, Heidelberg, Germany (2012)
- 3 Rountree et al. *Beyond DVFS: A First Look at Performance under a Hardware-Enforced Power Bound*, IPDPS Workshops, pp.947-953, IEEE Computer Society, (2012)
- 4 Wittman et al. *An Analysis of energy-optimized lattice-Boltzmann CFD simulations from the chip to the highly parallel level*, CoRR, abs/1304.7664 (2013)
- 5 Weaver V, Johnson M, Kasichayanula K, Ralph J, Luszczek P, Terpstra D, Moore S. *Measuring Energy and Power with PAPI* (2012), doi: 10.1109/ICPPW.2012.39

- 6 Browne S, Deane C, Ho G, Mucci P. In Proceedings of the Department of Defense HPCMP Users Group Conference, *PAPI: A Portable Interface to Hardware Performance Counters* (1999), pp. 7–10.
- 7 David H, Gorbato E, Hanebutte U, Khanna R, Le C. Low-Power Electronics and Design (ISLPED), *RAPL: memory power estimation and capping* (2010), pp. 189–194.
- 8 Intel Corp. 2012. *Intel 64 and IA-32 Architectures Software Developer Manual*.
- 9 *CPU Frequency Scaling*. Webpage:
https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling.
- 10 SuperMUC <https://www.lrz.de/services/compute/supermuc>
- 11 Intel Corp. 2012. Intel Xeon Processor. <http://www.intel.com/xeon>
- 12 Treibig J, Hager G, Wellein G. *LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments*, Proceedings of the 39th International Conference on Parallel Processing Workshops (2010), pp. 207–216, IEEE Computer Society, Washington, DC, USA (2010)
- 13 Treibig J, Hager G, Wellein G. *Best practices for HPM-assisted performance engineering on modern multicore processors*, Proceedings of the CoRR (2012)
- 14 Strohmaier E, Shan H. *Architecture independent performance characterization and benchmarking for scientific applications*. Proceedings of the The IEEE Computer Society’s 12th Annual International Symposium on Modeling, (2004), pp.467–474, IEEE Computer Society, Washington, DC, USA. (2004)
- 15 Strohmaier E, Shan H. *Apex-Map: A global data access benchmark to analyze HPC systems and parallel programming paradigms*, Proceedings of the SC2005 (2005). pp. 12–18
- 16 HWMON <https://www.kernel.org/doc/Documentation/hwmon/sysfs-interfaces>

3.25 High performance program generators: systematic construction with language support

Georg Ofenbeck (ETH Zürich, CH)

License © Creative Commons BY 3.0 Unported license
© Georg Ofenbeck

Main reference G. Ofenbeck, T. Rompf, A. Stojanov, M.n Odersky, M. Püschel, “Spiral in Scala: Towards the Systematic Construction of Generators for Performance Libraries,” in Proc. of the Int’l Conf. on Generative Programming: Concepts and Experiences (GPCE’13), pp. 125–134, ACM, 2013.


URL <http://dx.doi.org/10.1145/2517208.2517228>

Program generators for high performance libraries are an appealing solution to the recurring problem of porting and optimizing code with every new processor generation, but only few such generators exist to date. This is due to not only the difficulty of the design, but also of the actual implementation, which often results in an ad-hoc collection of standalone programs and scripts that are hard to extend, maintain, or reuse. In our recent work we asked the question which programming language concepts and features are needed to enable a more systematic construction of such generators.

This talk will provide a teaser on some of the solutions we investigated.

3.26 Methodology for MPI Applications Autotuning

Antonio Pimenta (Autonomus University of Barcelona, SP)

License  Creative Commons BY 3.0 Unported license
© Antonio Pimenta

We present a methodology designed to tackle the most common problems of MPI parallel programs. By developing a methodology that applies simple steps in a systematic way, we expect to obtain the basis for a successful autotuning approach of MPI applications based on measurements taken from their own execution. As part of the AutoTune project, our work is ultimately aimed at extending Periscope to apply automatic tuning to parallel applications and thus provide a straightforward way of tuning MPI parallel codes. Experimental tests demonstrate that this methodology could lead to significant performance improvements.

3.27 Exploiting processor inhomogeneity for performance optimization under a power bound


Barry Roundtree (LLNL – Livermore, US)

License  Creative Commons BY 3.0 Unported license
© Barry Roundtree

Cluster heterogeneity has traditionally referred to the presence of multiple different kinds of processors. However, significant variation in efficiency exists within individual processors families, with 10% variation observed in current processors and up to 25% expected in upcoming generations. While it is possible to mask these differences with intelligent power scheduling, the use of autotuning for performance optimization under a power bound will require exploiting these differences. In order to do so, however, we will have to rethink how jobs are scheduled, how MPI ranks (or equivalent) are mapped to processors, runtime critical path discovery as well as how supercomputers are designed and built.

3.28 Tuning support for a Xeon Phi template library

Martin Sandrieser (Universität Wien, AT)

License  Creative Commons BY 3.0 Unported license
© Martin Sandrieser
Joint work of Bajrovic, Enes; Benkner, Siegfried; Dokulil, Jiri; Sandrieser, Martin


We developed a template library that provides TBB-like parallel patterns that are executed in a hybrid fashion on the host and one or more Xeon Phi coprocessors simultaneously. The performance achievable with this library is profoundly influenced by the distribution of work among host processors and the coprocessors. Work distribution is performed automatically at runtime depending on different tunable parameters. These parameters can be reconfigured during runtime. In this talk we outline the runtime support implemented for our library, the initial support for autotuning, and discuss possible integration with existing autotuning frameworks.

References

- 1 J. Dokulil, E. Bajrovic, S. Benkner, M. Sandrieser, and B. Bachmayer. *HyPHI – task based hybrid execution C++ library for the Intel Xeon Phi coprocessor*. In 42nd International Conference on Parallel Processing (ICPP-2013), Lyon, France, 2013.

3.29 Performance Analysis, Energy Efficiency Optimizations and Auto-Tuning


Robert Schoene (TU Dresden, DE)

License  Creative Commons BY 3.0 Unported license
© Robert Schoene

With an objective target of 20 MW for an Exascale system, the power budget for future super computers has a severe restriction. Fortunately, current hardware and runtime environments feature various adjustable parameters that can be changed at runtime in order to confine the power usage. We present an overview of the potentials and limitations of several power saving mechanisms and describe the integration of energy efficiency optimizations in existing performance analysis tools. We also introduce an energy efficiency tuning cycle and exemplify its usage.

3.30 Providing the Necessary Semantic Context for Performance Tools and Autotuners

Martin Schulz (LLNL – Livermore, US)

License  Creative Commons BY 3.0 Unported license
© Martin Schulz

Performance tools and auto tuning systems alike require or can benefit from semantic context information as this enables a direct correlation and attribution of performance observations to application properties. In this talk I will show a few case studies in which we used such information to aid in the analysis of performance data. Further, I will introduce a system that can be used to create the necessary APIs for application programmers to safely express such semantic information.

3.31 Smarter Libraries with Synthesis

Armando Solar-Lezama (MIT, US)

License  Creative Commons BY 3.0 Unported license
© Armando Solar-Lezama

Performance tools and auto tuning systems alike require or can benefit from semantic context information as this enables a direct correlation and attribution of performance observations to application properties. In this talk I will show a few case studies in which we used such information to aid in the analysis of performance data. Further, I will introduce a system that can be used to create the necessary APIs for application programmers to safely express such semantic information.

3.32 Application-independent Autotuning for GPUs

Walter F. Tichy (*KIT – Karlsruhe Institute of Technology, DE*)

License © Creative Commons BY 3.0 Unported license
© Walter F. Tichy

We investigate the potential of online autotuning for general purpose computation on GPUs. Our application-independent autotuner AtuneRT optimizes GPU-specific parameters such as block size and loop-unrolling degree. We also discuss the peculiarities of auto-tuning on GPUs. We demonstrate tuning potential using CUDA and by instrumenting the parallel algorithms library Thrust. We evaluate our online autotuning approach on three GPUs with four applications.

3.33 Generalized roofline analysis?

Richard Vuduc (*Georgia Institute of Technology – Atlanta, US*)

License © Creative Commons BY 3.0 Unported license
© Richard Vuduc

Joint work of Choi, Jee; Vuduc, Richard

Main reference J.W. Choi, D. Bedard, R. Fowler, R. Vuduc. “A roofline model of energy,” in Proc. of the IEEE 27th Int’l Symp. on Parallel & Distributed Processing (IPDPS’13), pp. 661-672, IEEE CS, 2013.

URL <http://dx.doi.org/10.1109/IPDPS.2013.77>

The roofline model of Williams, Waterman, and Patterson (2009) is an analytic tool for gaining insights into how performance changes as intrinsic properties of a computation interact with features of a hardware system. We are generalizing the original roofline to develop a first-principles understanding the relationships among computational time, energy, and power. Our initial suggests how we might use these to explore the landscape of systems and algorithms that might be possible in the future.

References

- 1 Jee Whan Choi and Daniel Bedard and Robert Fowler and Richard Vuduc. “A roofline model of energy.” In *Proceedings of the IEEE Parallel and Distributed Processing Symposium (IPDPS)*, Boston, MA, USA, May 2013. DOI: 10.1109/IPDPS.2013.77
- 2 Kenneth Czechowski and Richard Vuduc. “A theoretical framework for algorithm-architecture co-design.” In *Proceedings of the IEEE Parallel and Distributed Processing Symposium (IPDPS)*, Boston, MA, USA, May 2013. DOI: 10.1109/IPDPS.2013.99

3.34 Autotuning for Scale

Felix Wolf (*German Research School for Simulation Sciences, DE*)

License © Creative Commons BY 3.0 Unported license
© Felix Wolf

Joint work of Calotoiu, Alexandru; Hoefler, Torsten; Poke, Marius; Felix Wolf

Main reference A. Calotoiu, T. Hoefler, M. Poke, F. Wolf, “Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes,” in Proc. of the Int’l Conf. for High Performance Computing, Networking, Storage and Analysis (SC’13), pp. 45:1–45:12, ACM, 2013.

URL <http://dx.doi.org/10.1145/2503210.2503277>

In this talk, we present a method designed to detect scalability bugs based on automatically generated performance models and discuss how it can be used to autotune for scale. Applic-

ations include the comparison of different algorithmic choices or execution configurations with respect to scalability. Since our method requires only small-scale experiments to make predictions for larger scales, the cost of this approach is relatively low even if the number of variants to be compared is high.

4 Working Groups

4.1 Energy and power autotuning

This breakout session investigated goals and techniques for energy and power optimization. While energy consumption reduction was only shortly discussed in the group of about 12 people, the main focus was on power issues in future exascale computers.

LLNL estimates that their first exascale system will be able to run with 60 MW but only 20 MW will be available to the machine. A first power budget limited machine is already under procurement at LLNL.

The contract with the energy company will be optimized for a steady use of the power. While it will have an upper limit of 20 MW, lower energy consumption will lead to higher costs as well. The motivation behind this is to enable the power company to optimize for a steady flow of energy. In addition, sudden drops in power usage will also not be possible to protect the power supply infrastructure from damage. These constraints require careful management of the power budget.

While a limitation of the power used by an application can be achieved by extended methods for power capping going beyond the processor's RAPL support. The second issue to not suddenly drop the machine's power consumption requires some additional mechanism, such as careful management by the job scheduler. In addition scenarios like trading power among the processes of an application to speedup those on the critical path as well as between different applications might gain importance on such machines.

The discussion on energy reduction concentrated on collecting the already applied techniques. Many projects explored the use of DVFS to reduce the energy by allowing to slow down the application for a limited percentage of the execution time. Instead of influencing the frequency, the same can be achieved by using the processor's power capping mechanism and letting the processor figure out how far to reduce the frequency.

Another approach is to trade performance for energy savings is to apply concurrency throttling. While selecting more efficient parallel execution with less threads or processes an energy reduction can be achieved.

A third approach is to slow down fast processors in case of load imbalances. Instead of a race to idle, these processors will arrive just in time while consuming less energy.

Besides reduction of the frequency and the voltage, switching off components might give significant power reduction. For example, the vector units of the processors are quite power hungry and should be switched off if not used efficiently.

In general, the group agreed that any performance tuning techniques are good for reduction of the energy, since faster and more efficient computation most frequently reduces the energy due to the constant static power used in the processors.

4.2 Language support for Autotuning

This breakout session discussed the issues of language support for autotuning. There was quite a bit of discussion about what it means to have language support for autotuning. Eventually we agreed there are two rather different approaches to languages for autotuning. One approach is to provide extensions to current languages to express tuning options in the form of parameters and algorithmic choices. The second approach is to develop new high level languages that support expressing the program to be performed in more abstract ways to allow an autotuner to generate many different program variants. The relative advantages of the two approaches are:

Extensions: the extension approach allows simpler integration with existing languages and would permit software to be evolved to be autotuned over time. These extensions could be developed in the form of pragmas that would be automatically ignored by compilers that didn't support autotuning or the specific pragma. Support could also be included in the extensions to describe hints about how the various parameters and pragmas interact. This could include both constraints such as one parameter must be less than the another to hints about the relative importance of different options.

New Languages: the new languages approach would allow for more aggressive autotuning. Programs would be written in a very high level language more like Matlab, and then compiled and autotuned to machine code. Machine specific descriptions of the transformations and the autotuning options could be written by people other than the application developers and potential reused for multiple programs.

4.3 Infrastructures

The breakout session about infrastructures consisted mainly of discussions related to what performance analysis and measurement tools can provide to automatic tuners. The consensus was that automatic tuners are stand-alone components that exist outside of the tools (with the possible exception of Periscope). What can be provided by tools (as well as applications) are operations that split logically into information and control related.

Information interfaces are already offered by several tools. In many cases, runtime environments also offer information interfaces. With the release of the MPI-3 standard, many MPI implementations give information about communication performance, message queue states, protocol thresholds, among others. Most tool developers seem to agree that exposing information that can be useful for automatic tuners is not only desirable, but also does not require a lot of development time.

In contrast with their posture regarding information interfaces, not all groups agreed on the feasibility or usefulness of control interfaces. The debate centered around the required scope of the automatic tuners. For example, some argued that the tuner should not touch internal parameters of communication libraries, such as MPI, while others saw the potential performance benefit achievable as worth pursuing. MPI does allow implementors to allow control access to internal parameters through MPIT, if they so desire.

A common issue that was discussed is that, because tools work at different abstraction levels and target hardware depending on what they measure or analyze, developing a common set of APIs for information and control will be difficult. Having a common set of APIs would allow an automatic tuner to interoperate with multiple tools in a modular manner.

Experienced members in the discussion shared that the tools community has not managed to come up with standard APIs and data structures even after many years of cooperation.

It was generally agreed that online access APIs to information and control APIs can greatly benefit automatic tuners. The need for restarts increases the time required for tuning significantly. Together with online approaches, modeling can be used to reduce the times required while optimizing. It was also agreed that, depending on the frequency of samples and scale of applications, the balance between flexibility and processing time required for these online interfaces may differ.

4.4 Black/Whitebox Autotuning

In this breakout session we discussed various approaches to autotuning that we classified as black box and white box.

Black Box: In black box approaches the autotuning search machinery does not have knowledge of the search space and thus cannot employ any knowledge about the underlying problem, except for knowledge obtained through evaluating instance. Notable black box successes include ActiveHarmony+Chill and profile guided optimization in compilers like the Intel compiler and the IBM XL compiler.

White Box: In white box approaches the search machinery has some understanding of the underlying problem structure and thus the search can be guided. Notable successes in white box autotuning include ATLAS and Spiral.

Generally, black box approaches are wider applicable as they do not need to understand the underlying problem structure, while white box approaches are limited to well-understood kernels or libraries that are easier to model.

Other aspects that separate the approaches are the cost to evaluate search points (re-compile versus relaunch), model quality and constraints in guided search. Model types and outputs are covering a wide range of possibilities and models may not be very accurate by nature. Types of models may vary, they may include target architecture, operating systems, and data sets or even the semantics of the tuned program. A model may be predictive or just provide relative ordering of instances. The working group felt that a “trust but verify” approach to models is needed in autotuning.

4.5 Search Algorithms

During this breakout session the group brought together an number of different search algorithms that are used in autotuning systems. Autotuning starts from a search space that is constructed from valid setting for the tuning parameters. The following search algorithms to walk the search space are available:

Exhaustive: The full search space is explored and all the combinations of the tuning parameters are evaluated.

Random: This algorithm selects valid points in the search space randomly. Random and exhaustive are good for autotuning if nothing is known about the structure of the objective function.

Meta-heuristics: This class covers genetic algorithms and simulated annealing.

Derivative-free algorithms: To this class belong algorithms such as parallel rank order, nelder mead, hill climbing, and tabu search.

Machine learning: Several techniques from machine learning can be applied to improve autotuning, such as neural networks, decision trees, and support vector machines. All these techniques try to learn a correlation between an application signature and the tuning parameters or the application performance. They can be applied to predict the application performance for certain parameter settings or as an oracle to predict the best or a best set of parameter settings. An important program with machine learning techniques is to find the right set of application properties that discriminate the applications.

Rule-based: Another way to shrink the search space is to apply rules that let the autotuner deduce a certain subspace.

Besides the collection of search algorithms, the group discussed the aspect of generating valid points in the search space when a number of constraints are given. Several tools have used Presburger Arithmetic to solve this problem.

In important aspect in autotuning is also the tradeoff between quality of the results and the effort spent in searching for the best solution. Some existing tools provide a search time limit and return the found best solution when the limit is reached. This is especially used for meta-heuristics and random search.

Besides using just one search algorithms, multiple algorithms can be combined to solve the problem. A good example is to use machine learning to get a good estimate and perform a local search afterwards. Also a random selection of starting points with a local search might be useful.

At the end, the group also discussed how the quality of search algorithms can be evaluated. A comparison with exhaustive search is typically due to the size of the search space impossible. One approach is to replace exhaustive with random or to compare different algorithms by selecting the same number of tries for the algorithms.

Participants

- Enes Bajrovic
Universität Wien, AT
- Shajulin Benedict
St.Xavier's Catholic College of
Engineering, IN
- Siegfried Benkner
Universität Wien, AT
- Aydın Buluç
Lawrence Berkeley National
Laboratory, US
- Milind Chabbi
Rice University, US
- I-hsin Chung
IBM TJ Watson Res. Center –
Yorktown Heights, US
- Isaias Alberto Compres Urena
TU München, DE
- Guojing Cong
IBM TJ Watson Res. Center –
Yorktown Heights, US
- Thomas Fahringer
Universität Innsbruck, AT
- Franz Franchetti
Carnegie Mellon University, US
- Grigori Fursin
INRIA Saclay – Île-de-France –
Orsay, FR
- Hans Michael Gerndt
TU München, DE
- Carla Guillen
Leibniz Rechenzentrum –
München, DE
- Torsten Höfler
ETH Zürich, CH
- Jeffrey K. Hollingsworth
University of Maryland –
College Park, US
- Paul D. Hovland
Argonne National Laboratory, US
- Toshiyuki Imamura
RIKEN – Kobe, JP
- Thomas Karcher
KIT – Karlsruhe Institute of
Technology, DE
- Takahiro Katagiri
University of Tokyo, JP
- Michael Knobloch
Jülich Supercomputing
Centre, DE
- Andreas Knüpfer
TU Dresden, DE
- Jakub Kurzak
University of Tennessee, US
- Allen D. Malony
University of Oregon, US
- Andrea Martinez
Autonomous University of
Barcelona, ES
- Renato Miceli Costa Ribeiro
ICHEC – Galway, IE
- Robert Mijakovic
TU München, DE
- Bernd Mohr
Jülich Supercomputing
Centre, DE
- Shirley V. Moore
University of Texas – El Paso, US
- Carmen Novarrete
Leibniz Rechenzentrum –
München, DE
- Georg Ofenbeck
ETH Zürich, CH
- Antonio Pimenta
Autonomous University of
Barcelona, ES
- Barry Rountree
LLNL – Livermore, US
- Martin Sandrieser
Universität Wien, AT
- Robert Schoene
TU Dresden, DE
- Martin Schulz
LLNL – Livermore, US
- Armando Solar-Lezama
MIT, US
- Walter F. Tichy
KIT – Karlsruhe Institute of
Technology, DE
- Jesper Larsson Träff
TU Wien, AT
- Richard M. Veras
Carnegie Mellon University, US
- Richard Vuduc
Georgia Institute of Technology –
Atlanta, US
- Felix Wolf
German Research School for
Simulation Sciences, DE

