Report from Dagstuhl Seminar 17431

Performance Portability in Extreme Scale Computing: Metrics, Challenges, Solutions

Edited by

Anshu Dubey[1], Paul H. J. Kelly[2], Bernd Mohr[3], and Jeffrey S. Vetter[4]

1 **Argonne National Laboratory, US,** adubey@anl.gov
2 **Imperial College London, GB,** p.kelly@imperial.ac.uk
3 **Jülich Supercomputing Centre, DE,** b.mohr@fz-juelich.de
4 **Oak Ridge National Laboratory, US,** vetter@ornl.gov

Abstract

This Dagstuhl Seminar represented a unique opportunity to bring together international experts from the three research communities essential to tackling the HPC performance portability challenge: developers of large-scale computational science software projects, researchers developing parallel programming technologies, and performance specialists. The major research questions for the seminar were to understand challenges, design metrics, and prioritize potential solutions for performance portability, management of data movement in complex applications, composability, and pathways to impact on the research community. The overall conclusion shared by all participants was that performance portability in extreme scale computing can be achieved, especially if parallel applications are designed with performance portability in mind from the beginning. Making legacy application performance portable still requires enormous efforts and expertise. In many instances it will likely require extensive refactoring.

Seminar October 22–27, 2017 – http://www.dagstuhl.de/17431
1998 ACM Subject Classification Parallel architectures, Parallel programming languages, Parallel algorithms, Performance,
Keywords and phrases Parallel programming, performance portability, productivity, scientific computing
Digital Object Identifier 10.4230/DagRep.7.10.84

1 Executive Summary

Anshu Dubey
Paul H. J. Kelly
Bernd Mohr
Jeffrey S. Vetter

This report documents the program and the outcomes of Dagstuhl Seminar 17431 "Performance Portability in Extreme Scale Computing: Metrics, Challenges, Solutions".

Performance Portability is a critical new challenge in extreme-scale computing. In essence, performance-portable applications can be efficiently executed on a wide variety of HPC architectures without significant manual modifications. For nearly two decades, HPC architectures and programming models remained relatively stable, which allowed growth of

Performance Portability in Extreme Scale Computing: Metrics, Challenges, Solutions, *Dagstuhl Reports*, Vol. 7, Issue 10, pp. 84–110
Editors: Anshu Dubey, Paul H. J. Kelly, Bernd Mohr, and Jeffrey S. Vetter

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

complex multidisciplinary applications whose lifecycles span multiple generations of HPC platforms.

Recently, however, platforms are growing much more complex, diverse, and heterogeneous - both within a single system and across systems and generations. Details already known from planned future systems indicate that this trend will continue (at least for the foreseeable future). Current and planned future large-scale HPC systems consist of complex configurations with a massive number of components. Each node has multiple multi-core sockets and often one or more additional accelerator units in the form of many-core nodes or GPGPUs, resulting in a heterogeneous system architecture. Memory hierarchies including caches, memory, and storage are also diversifying in order to meet multiple constraints: power, latency, bandwidth, persistence, reliability, and capacity. These factors are reducing portability, and forcing applications teams to either spend considerable effort porting and optimizing their applications for each specific platform, or risk owning applications that perform well on perhaps only one architecture. The latter option would still require porting and optimizing effort for each new generation of systems.

This Dagstuhl Seminar represented a unique opportunity to bring together international experts from the three research communities essential to tackling this performance portability challenge: developers of large-scale computational science software projects whose lifetime will span multiple generations of systems, researchers developing relevant parallel programming or system software technologies, and specialists in profiling, understanding, and modelling performance. The major research questions for the seminar were:

- To understand challenges, design metrics, and prioritize potential solutions for performance portability: Solutions will need to synthesize existing concepts across multiple fields, including performance and productivity modeling, programming models and compilation, architectures, system software.
- Management of data movement in complex applications: Diverse data movement patterns dictated by different devices form one of the largest impediments to portable performance. Addressing it will require cross-cutting solutions supporting more than one abstraction, and will allow scientists to balance tradeoffs in these factors prior to design, development, or procurement of an architecture, software stack, or application.
- Composability: Many applications require flexibility and composability because they address different physical regimes either within the same simulation, or in different instances of simulations.
- Pathways to impact on the research community: As the community becomes more reliant on both more complex architectures and software stacks, it is especially important that we develop the conceptual tools to facilitate research and practical solutions for performance portability. The impact of ignoring this topic could be potentially devastating to the quality and sustainability of computational science software, and consequently on the science and engineering research they support. Thus a key element of the seminar will be to tackle this challenge in major science community software projects.

The seminar started with a series of flash talks, where participants introduced themselves in a two-minute one-slide presentation summarizing their contribution or interest in the seminar by providing two to three bullet points on (i) Challenge/Opportunity (WHY?) (ii) Timeliness (WHY NOW?) (iii) Approaches (HOW?) and (iv) IMPACT (SO WHAT?). Each day started with a longer keynote presentation by a representative of one of the major stakeholders in the field, followed by short presentations by participants grouped in sessions with a common relevant theme. Each keynote or short talk session ended with an extensive question-and-answer session and open discussion slot in which all the speakers from the session took part.

The overall conclusion shared by all participants was that performance portability in extreme scale computing can be achieved, especially if parallel applications are designed with performance portability in mind from the beginning. Model complexity and performance portability both require that frameworks be designed with composable components incorporating layers of abstraction so that trade-offs can be reasoned about. Making legacy application performance portable still requires enormous efforts and expertise. In many instances it will likely require extensive refactoring. Similar design principles regarding formulation of a flexible and composable framework apply for legacy software refactoring, along with strong emphasis on rigorous verification built into the process. The seminar recognized the challenges faced by the applications in adopting abstractions; converting research prototypes to reliable production-grade product. The adverse structure of incentives for both applications and abstractions, and the complexity of formulating a process or collaboration between the two communities, may be bigger barriers than technical challenges in making performance portability feasible. It is critical that the involved communities and stakeholders are made aware of these challenges while seeking solutions for sustainable computational science projects.

2 Table of Contents

3 Overview of Talks

3.1 Automatic Detection of Large Extended Data-Race-Free Regions with Conflict Isolation

Alexandra Jimborean (Uppsala University, SE)

Joint work of Alexandra Jimborean, Jonatan Waern, Per Ekemark, Stefanos Kaxiras, Alberto Ros
Main reference Alexandra Jimborean, Per Ekemark, Jonatan Waern, Stefanos Kaxiras, Alberto Ros: "Automatic Detection of Large Extended Data-Race-Free Regions with Conflict Isolation", IEEE Trans. Parallel Distrib. Syst., Vol. 29(3), pp. 527–541, 2018.
URL http://dx.doi.org/10.1109/TPDS.2017.2771509

Data-race-free (DRF) parallel programming becomes a standard as newly adopted memory models of mainstream programming languages such as C++ or Java impose data-race-freedom as a requirement. We propose compiler techniques that automatically delineate extended data-race-free (xDRF) regions, namely regions of code that provide the same guarantees as the synchronization-free regions (in the context of DRF codes). xDRF regions stretch across synchronization boundaries, function calls and loop back-edges and preserve the data-race-free semantics, thus increasing the optimization opportunities exposed to the compiler and to the underlying architecture. We further enlarge xDRF regions with a conflict isolation (CI) technique, delineating what we call xDRF-CI regions while preserving the same properties as xDRF regions. Our compiler (1) precisely analyzes the threads' memory accessing behavior and data sharing in shared-memory, general-purpose parallel applications, (2) isolates data-sharing and (3) marks the limits of xDRF-CI code regions. The contribution of this work consists in a simple but effective method to alleviate the drawbacks of the compiler's conservative nature in order to be competitive with (and even surpass) an expert in delineating xDRF regions manually. We evaluate the potential of our technique by employing xDRF and xDRF-CI region classification in a state-of-the-art, dual-mode cache coherence protocol. We show that xDRF regions reduce the coherence bookkeeping and enable optimizations for performance (6.4 percent) and energy efficiency (12.2 percent) compared to a standard directory-based coherence protocol. Enhancing the xDRF analysis with the conflict isolation technique improves performance by 7.1 percent and energy efficiency by 15.9 percent.

3.2 Is it performance portability when I'm using DGEMM?

Michael Bader (TU München, DE)

The earthquake simulation software SeisSol uses a high-order discontinuous Galerkin scheme for discretisation of the seismic wave equations. The scheme is formulated via small element-local matrix chain products; respective matrices include the matrix of quantities, stiffness matrices, discrete Jacobians, etc. As matrices can be sparse or dense, the matrix chain products are analysed for sparsity patterns of matrices, including intermediate and result matrices. The LIBXSMM library is used to generated high-performance code on Intel architectures.

While the matrix notation provides a suitable abstraction layer for expressing the numerical scheme, only a limited level of performance portability is reached by this approach. The presentation's goal was to discuss the current status and possible routes for extension and improvement.

References

1 A. Heinecke, G. Henry, M. Hutchinson and H. Pabst: LIBXSMM: Accelerating Small Matrix Multiplications by Runtime Code Generation, SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. DOI: 10.1109/SC.2016.83

2 C. Uphoff and M. Bader: Generating high performance matrix kernels for earthquake simulations with viscoelastic attenuation. In W. W. Smari (ed.), Proceedings of the 2016 International Conference on High Performance Computing & Simulation (HPCS 2016), p. 908–916. IEEE, 2016.

3 C. Uphoff, S. Rettenberger, M. Bader, E. H. Madden, T. Ulrich, S. Wollherr and A.-A. Gabriel: Extreme Scale Multi-Physics Simulations of the Tsunamigenic 2004 Sumatra Megathrust Earthquake. SC '17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2017

3.3 Performance Portability: Discussion Entry Points from Experience in OpenMP

Carlo Bertolli (IBM TJ Watson Research Center – Yorktown Heights, US)

In this talk I will provide a set of entry points for discussion related to performance portability challenges in the implementation of OpenMP on GPU-enabled systems. I will give an overview of how some of these challenges are being addressed by the OpenMP committee and what the main hard problems with proposed solutions are. I will also show how the performance portability issue spans beyond the traditional HPC community and is present in emerging fields such as cognitive computing and related hardware acceleration technology.

3.4 Portability of Performance in Generic Code

Mauro Bianco (CSCS – Lugano, CH)

Joint work of Mauro Bianco, Oliver Fuhrer, Carlos Osuna, Thomas Schulthess, Hannes Vogt

Generic Libraries can offer separation of concerns between program developers and performance specialists by providing abstractions for algorithmic motifs, so as to hide low-level details. As computing platforms diversify, the design of these libraries becomes more and more complex and more and more targeted to specific application fields. There are however some deep problems that make portability of performance extremely challenging in real-world applications. The effect is the tendency to limit the applicability of particular generic interfaces to sub-sets of problems in the targeted application fields, thus making them less and less general. By analyzing some of the characteristics of weather and climate applications,

this talk will highlight some challenges in the development of a truly performance-portable layer for this class of applications. Some of these challenges raise more general questions about performance in HPC. The talk focuses on the engineering challenges, such as development complexity, maintainability and interoperability with other languages. The result of the development is a set of libraries for production applications for weather and climate simulations to be used by different institutions.

3.5 Automatic Empirical Performance Modeling

Alexandru Calotoiu (TU Darmstadt, DE)

Given the tremendous cost of an exascale system, its architecture must match the requirements of the applications it is supposed to run as precisely as possible. Conversely, applications must be designed such that building an appropriate system becomes feasible, motivating the idea of co-design. In this process, a fundamental aspect of the application requirements are the rates at which the demand for different resources grows as a code is scaled to a larger machine. However, if the anticipated scale exceeds the size of available platforms this demand can no longer be measured. This is clearly the case when designing an exascale system. Moreover, creating analytical models to predict these requirements is often too laborious - especially when the number and complexity of target applications is high. We discuss how automated performance modeling can be used to quickly predict application requirements for varying scales and problem sizes. Following this approach, we show how to determine the exascale requirements of a code and use them to illustrate system design tradeoffs.

3.6 If the HPC Community were to create a truly productive language...[how] would we ever know?

Bradford Chamberlain (Cray Inc. – Seattle, US)

In this talk, I strived to explore the relationship between productivity and performance portability, particularly in the context of programming languages for High Performance Computing. I began with a quick argument that we don't really have productive languages in HPC, and went on to characterize currently used HPC "languages" (broadly interpreted) and what I imagine from productive languages. From there, I gave a brief, personal historical perspective on the effort to define metrics for measuring productivity under the DARPA HPCS program (2002-2012). I wrapped up with an argument that I believe productivity is a highly personal, social choice and that, for that reason, our evaluations of productivity should be more personal and social, supporting subjective decisions rather than analytic. I introduced the (mainstream) Computer Language Benchmarks Game as an instance of measuring properties of languages through programs written in those languages, and one supporting interactions with the results and programs in various ways. This led to my first suggestion which is that the HPC community would benefit from a similar arena. From there, I turned to a brief description of Chapel, highlighting key policies that we have opted

not to bake into the language in order to support productivity, a separation of concerns, and performance portability: user-defined parallel iterators, domain maps for implementing dense/sparse/associative domains & arrays, and locale models for representing the target architecture. I then posed the question as to how languages like Chapel can advance to the stage of being used "in HPC production," suggesting that opportunities (like a Dagstuhl seminar?) should be created for applications and languages people to pair program, cutting past superficial familiarity with each others' technologies.

3.7 Should I port my code to a DSL?

Aparna Chandramowlishwaran (University of California – Irvine, US)

Joint work of Aparna Chandramowlishwaran, Laleh Aghababaie Beni, Bahareh Mostafazadeh, Ferran Marti
Main reference Laleh Aghababaie Beni, Aparna Chandramowlishwaran: "PASCAL: A Parallel Algorithmic SCALable Framework for N-body Problems", in Proc. of the Euro-Par 2017: Parallel Processing - 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28 - September 1, 2017, Proceedings, Lecture Notes in Computer Science, Vol. 10417, pp. 482–496, Springer, 2017.
URL http://dx.doi.org/10.1007/978-3-319-64203-1_35

In this talk, I'll outline the key challenges in developing parallel algorithms and software for two classes of applications – N-body solvers and structured grid finite volume solvers on current and future platforms. Our goal is to reduce the apparent gap in performance between code generated from high-level forms and that of hand-tuned code, which we address using extensive characterization of the optimization space for these computations and automating the process through domain-specific languages (DSLs) and code generators. These application-specific compilers provide the domain scientists the ability to productively harness the power of these large machines and to enable large-scale scientific simulations and big data applications. However, the performance of DSLs has been a concern. Therefore, we specifically ask whether CFD and N-body applications expressed in these DSLs can deliver a sufficient combination of optimizations to compete with a hand-tuned code and what are its limitations.

3.8 Exascale Scientific Computing : The Road Ahead

Kemal A. Delic (Hewlett Packard – Grenoble, FR) and Martin A. Walker

Theory and experimentation have been the hallmark of scientific inquiries for centuries, while we observe the rising importance of simulation as the third, important pillar of scientific explorations. The key driving factor is slowly evolving field of hyperscale computing, recently augmented and accelerated through the confluence of artificial intelligence(AI), big data (BD) and the internet of things (IoT), making it again a leading force in scientific and industrial computing.

For the new scientific discoveries to happen, larger scale as well as different computing infrastructure is required to enable and support bigger data collections and more efficient/effective algorithms.

The rise of cloud computing,as a hyperscale computing infrastructure, has provided a new arena for the execution of scientific workloads. Such an infrastructure has also enabled great strides in the application of multilayered weighted networks often known as "neural networks" for scientific enquiry. The universal approximation theorem, implies that such networks can be constructed to solve partial differential equations, and therefore applied to computing the consequences of scientific theories.

Construction of such networks ("training", i.e. the determination of the number of nodes and topology of the network, together with the values of the weights on each edge, and the choice of trigger function at the nodes) requires extensive computing with large data volumes on very large infrastructure until a sufficiently accurate model has been achieved. The resulting model can then be executed ("inference"), either on general purpose or specialized hardware accelerators. We believe that critical applications such as high-frequency trading (HFT), self-driving cars and various drones, as wells as edge network devices will require both: special hybrid architectures combining general purpose CPUs and specialized GPU, TPU, FPGA for training, and special chips for on-board execution.

In conclusion, looking bravely into next 50 years, we see High Performance Computing being augmented and advanced with AI, BD and IoT, evolving into Neuromorphic Computing within the next 15 years and with Quantum Computing on a 50-year horizon.

This is an exciting opportunity for emerging generations of scientists advancing scientific knowledge using new types of scientific instruments based on advances in computing sciences and clever engineering, producing unprecedented exascale scientific machines. It is our high hope that they will augment and accelerate the pace of scientific inquiries. Emerging Exascale Computing Systems may enable advances similar to the scientific advances realized by the invention of both the microscope and telescope a few centuries ago.

3.9 Beyond algorithmic patterns: tackling the performance portability challenge with hardware paradigm optimisation patterns

Christophe Dubach (University of Edinburgh, GB)

Algorithmic patterns have emerged as a solution to exploit parallel hardware. Applications are expressed at a high-level, using a small set of well known patterns of code adapted to each application domain. This approach hides the hardware complexity away from programmers and shifts the responsibility for extracting performance to the library writer or compiler. However, producing efficient implementations remains a complicated task that needs to be repeated for each newly introduced high-level pattern or whenever hardware changes.

The first part of the presentation will show how typical optimisations performed by GPU programmers are expressible as optimisation patterns and how an optimisation space can be defined in terms of provably correct rewrite-rules. Our initial results show that this approach leads to performance portability on several classes of GPUs.

The second part of the talk will focus on showing how optimisation patterns could be generalised to a larger classes of hardware including CPUs, GPUs, FPGAs and multi-node clusters. The main idea is to develop an abstraction to reasons about hardware paradigms (e.g. memory hierarchy, parallelism hierarchy, synchronisation primitives, communication primitives) and a set of corresponding hardware-specific optimisation patterns. This would

enable the automatic generation of an optimising code generator tailored for a particular instance of a heterogeneous parallel machine (e.g. cluster of GPUs or FPGAs) using a high-level description of the machine.

3.10 Kokkos: Performance Portability and Productivity for C++ Applications

H. Carter Edwards (Sandia National Labs – Albuquerque, US)

Joint work of H. Carter Edwards, Christian Trott, Daniel Sunderland, Daniel Ibanez, Nathan Ellingwood
Main reference H. Carter Edwards, Christian R. Trott, Daniel Sunderland: "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns", J. Parallel Distrib. Comput., Vol. 74(12), pp. 3202–3216, 2014.
URL http://dx.doi.org/10.1016/j.jpdc.2014.07.003

Summary presentation of the Kokkos programming model and C++ library implementation for performance portability and productivity of intra-node parallel computations across diverse multicore and manycore architectures. This summary includes the "1+epsilon" versions of application code for performance portability, and the programming model abstractions enabling Kokkos to achieve this goal. An overview is given for Kokkos' data parallel and directed acyclic graph of tasks (task-dag) patterns to illustrate how Kokkos enables application development productivity for intra-node parallel algorithms.

3.11 How to define upper performance bounds using analytic performance models – Opportunities and Limitations

Jan Eitzinger (Universität Erlangen-Nürnberg, DE) and Georg Hager

Main reference Georg Hager, Jan Treibig, Johannes Habich, Gerhard Wellein: "Exploring performance and power properties of modern multi-core chips via simple machine models", Concurrency and Computation: Practice and Experience, Vol. 28(2), pp. 189–210, 2016.
URL http://dx.doi.org/10.1002/cpe.3180

Talking about application performance on computer systems ideally requires to define an analytic upper performance bound. Performance is defined by how a specific software interacts with the machine, which is the processor, its memory hierarchy and external IO devices as persistent storage and network. Many publications judge about 'good' or 'bad' performance in the light of comparisons to other implementations or hardware platforms. Still the insight created by such statements is usually very small. Automatic machine learning approaches fulfil certain purposes for predicting performance or are used to extrapolate measurements. But are those methods generating a deeper understanding about bottlenecks and optimisation opportunities? This talks tries to ignite a discussion about if and how (preferably simple) analytic performance modelling can help to make sense of an observed performance number and where its limitations are.

3.12 Deploying performance portable code

Todd Gamblin (LLNL – Livermore, US)

URL https://computation.llnl.gov/projects/spack-hpc-package-manager

Performance portability is a hot research topic, but are code developers really striving for it? HPC application teams are mainly tasked with producing good science, and ensuring performance portability takes a lot of time. Application teams fighting to make deadlines do some basic optimizations, but the risk/reward ratio for more sophisticated techniques is still far too high. Current performance portability techniques require significant effort on the part of the installer or the application developer. Most people installing HPC software are still building from source, from scratch, with compilers the code developer may or may not have tested with. The build is hard enough, even without considering a potential tuning phase. Package managers provide a natural harness around source builds of HPC code. However, most package managers in wide distribution sacrifice performance for potability and assume lowest-common-denominator flags.

In this talk, we discuss Spack, a package manager for HPC, and how the Spack community is looking to address performance portability at the software distribution level. We look at how performance portability can affect software source and binary distributions, and what type of additional infrastructure and tooling we would need to distribute tuned, optimized software for all HPC users. Our hope is to one day democratize performance portability.

3.13 Porting Atmosphere Kernels on Various System

Lin Gan (Tsinghua University – Beijing, CN)

A summary of previous work on porting atmosphere kernels on different platform, including the Sunway TaihuLight. Performance portability of atmosphere code is bad, so efforts have to be made and patience is required. For Sunway system, different software is being developed to make it easy for application to be ported.

3.14 Performance, Portability, and Dreams

William D. Gropp (University of Illinois – Urbana-Champaign, US)

URL http://wgropp.cs.illinois.edu/bib/talks/tdata/2017/dagstuhl-keynote.pdf

Why do we care about performance portability? A major reason is because a big part of the programming crisis is caused by the challenge of obtaining performance on even a single platform. And achieving performance is hard - systems are complex, behavior has random elements, and the behavior of interactions between parts is hard to predict. And after more than 20 years of relative architectural stability, processor architecture is diversifying, making the problem even worse.

But performance portability itself is not an absolute goal. Implicitly, performance portability is intended to reduce the amount of work needed to achieve adequate performance. How much programmer rework is acceptable to achieve performance portability? What other limitations, such as code complexity or sensitivity to input data, are acceptable? And there are dangers to making performance portability the goal. For example, one way to achieve performance portability is to make all performance mediocre. Then performance is similar on all platforms, but nowhere good. A good definition for performance portability is clearly necessary, but a workable definition is quite difficult. Most current definitions are either very difficult to apply (because they refer to a hard-to-determine theoretical achievable performance) or are susceptible to odd effects (when based on some specific code and that code's performance; leading, for example, to the case where any single code is performance portable by definition until a second code is created).

There are many different approaches to performance portability. These include enhancements to existing languages, new programming languages, libraries, tools, and even general techniques. The presentation provides a few examples that show that even for seemingly simple examples, performance is difficult to achieve without exploiting information known only at runtime. This suggests that approaches to performance portability need to include ways to adapt, perhaps at runtime, to different input data and different system behavior.

We discuss the Illinois Coding Environment (ICE), an example of an approach that uses annotations to an existing language to provide additional information that an guide performance optimizations, and uses a framework that can invoke third-party tools to apply performance enhancing transformations.

All approaches that rely on transformations to the user's code must address the issue of correctness - ensuring that any transformations do not introduce errors into the code. We point out that it is necessary to prove such transformations are correct, but that is not sufficient, because correctness requires that the entire system (including low-level software and all hardware) also correctly execute the transformed code.

The presentation ends by arguing that rather than try to define what performance portability is, the community should focus on the goals - making it easier for end users to run an application code effectively on different systems, and making it easier for developers to write, tune, and maintain an application for multiple systems.

3.15 Performance Portability Using Compiler-Directed Autotuning

Mary W. Hall (University of Utah – Salt Lake City, US)

As current and future architectures become increasingly diverse, the challenges of developing high-performance applications are becoming more onerous. The goal of compiler optimization in high-performance computing is to take as input a computation that is architecture independent and maintainable and produce as output efficient implementations of the computation that are specialized for the target architecture. A compiler that is specialized for an application domain can tailor its optimization strategy to increase effectiveness. This talk describes how domain-specific optimizations can be combined with standard polyhedral compiler transformation and code generation technology to achieve very high levels of performance, comparable to what is obtained manually by experts. Polyhedral

frameworks permit composition of complex transformation sequences with robust code generation. Autotuning empirically evaluates a search space of possible implementations of a computation to identify the implementation that best meets its optimization criteria (e.g., performance, power, or both). Combining the three concepts, autotuning compilers generate this search space of highly-tuned implementations either automatically or with programmer guidance. We describe the application of this approach to three domains: geometric multigrid and the stencil computations within them, tensor contractions and sparse matrix computations.

3.16 User Interfaces to Performance: Kernel Transformation with Loopy

Andreas Klöckner (University of Illinois – Urbana-Champaign, US) and Matt Wala

Focusing on the optimization of computational kernels, Loopy is a transformation-based programming system embedded in Python that aims to assist with all stages of the performance engineering process from within a single, user-exposed intermediate representation (IR). This single IR is based on scalar assignemnt and polyhedrally-specified control flow. It is designed to easily capture code at many levels of abstraction, ranging from high-level, mathematical formulas to machine concerns such as vectorization, memory access patterns, parallelization, ILP, and many more. A large cross-section of tuning concerns are amenable to modification by transformations acting on the IR. The IR is capable of capturing data-dependent control flow, global barrier synchronization (compiled to multiple GPU kernels for compatibility), reductions, and prefix sums. In addition, the IR lends itself to counting and modeling computational expense, enabling manual and automated tuning. Multiple targets allow code generation for CPUs, Intel Knights machines as well as GPUs. A live demonstration of these capabilities will be part of this presentation.

Applications for which Loopy has been demonstrated to achieve good performance across architectures include high-order finite elements, chemical kinetics, and numerical linear algebra.

3.17 CnC for future-proof application development

Kathleen Knobe (Rice University – Houston, US)

The history of computing shows us that we have limited ability to predict important architectural features or important optimization concerns too far in the future. The applications that might have to be ported due to these changes are growing rapidly in number, size and complexity. The cost of re-implementing an app or a suite of apps for each new architectural advance or each new optimization goal is exorbitant.

Current approaches that address a range of foreseeable architectures and concerns are very helpful. CnC is, instead, an approach that addresses the fact that there will be unforeseen architectures with their unforeseen optimizations concerns.

The basic approach is one of separation of concerns. This is critical

- Not only for predicted architectures but for unpredicted architectures
- Not only for predicted optimization goals but for unpredicted system styles and unpredicted optimization goals.

The basic idea is to develop a program description that includes

- Everything about the meaning of the program
- Everything that might be useful for optimization

but it explicitly does not include anything that might have to be undone to improve performance for some new, unforeseen architecture or optimization goal. This application description is then paired with an appropriate tuning and/or runtime, and even a new approach to tuning and/or runtime.

The initial motivation for CnC was to support the separation of concerns between the activities of the domain-expert from those of the tuning-expert even within an individual but also among distinct professionals, each with their own expertise. This isolation now supports an unchanged application specification with a wide variety of specific tunings and even a variety of runtime styles. We believe it inherently supports new, unpredicted architectures and tuning goals. We also believe that a variety of current approaches to performance portability would pair well with this style of application specification.

The talk will describe the CnC program description language showing the rationale for its features. It will then describe briefly a wide variety of runtime systems and optimizations that have already been implemented as well as some that we're planning.

3.18 Performance Portability Challenges in FPGAs

Naoya Maruyama (LLNL – Livermore, US)

This talk will discuss performance portability challenges when using FPGAs as an accelerator. Developing applications that run portability across devices including FPGAs is now feasible by using common programming interfaces such as OpenCL. However, exploiting performance of FPGAs tends to require FPGA-specific parallelization and optimization, causing performance portability challenges. We will show several case studies using OpenCL-based benchmarks.

3.19 Crossing the Chasm: How to develop weather and climate models for next generation computers?

Chris Maynard (MetOffice – Exeter, GB)

Joint work of Bryan N. Lawrence, Michael Rezny, Reinhard Budich, Peter Bauer, Jörg Behrens, Mick Carter, Willem Deconinck, Rupert Ford, Christopher Maynard, Steven Mullerworth, Carlos Osuna, Andrew Porter, Kim Serradell, Sophie Valcke, Nils Wedi, Simon Wilson
Main reference Bryan N. Lawrence, Michael Rezny, Reinhard Budich, Peter Bauer, Jörg Behrens, Mick Carter, Willem Deconinck, Rupert Ford, Christopher Maynard, Steven Mullerworth, Carlos Osuna, Andrew Porter, Kim Serradell, Sophie Valcke, Nils Wedi, Simon Wilson: "Crossing the Chasm: How to develop weather and climate models for next generation computers?", Geosci. Model Dev. Discuss., in review, 2017.
URL https://doi.org/10.5194/gmd-2017-186

Weather and climate models are complex pieces of software which include many individual components, each of which is evolving under the pressure to exploit advances in computing to enhance some combination of a range of possible improvements (higher spatio/temporal resolution, increased fidelity in terms of resolved processes, more quantification of uncertainty etc). However, after many years of a relatively stable computing environment with little choice in processing architecture or programming paradigm (basically X86 processors using MPI for parallelism), the existing menu of processor choices includes significant diversity, and more is on the horizon. This computational diversity, coupled with ever increasing software complexity, leads to the very real possibility that weather and climate modelling will arrive at a chasm which will separate scientific aspiration from our ability to develop and/or rapidly adapt codes to the available hardware.

In this paper we review the hardware and software trends which are leading us towards this chasm, before describing current progress in addressing some of the tools which we may be able to use to bridge the chasm. This brief introduction to current tools and plans is followed by a discussion outlining the scientific requirements for quality model codes which have satisfactory performance and portability, while simultaneously supporting productive scientific evolution. We assert that the existing method of incremental model improvements employing small steps which adjust to the changing hardware environment is likely to be inadequate for crossing the chasm between aspiration and hardware at a satisfactory pace, in part because institutions cannot have all the relevant expertise in house. Instead, we outline a methodology based on large community efforts in engineering and standardisation, one which will depend on identifying a taxonomy of key activities – perhaps based on existing efforts to develop domain specific languages, identify common patterns in weather and climate codes, and develop community approaches to commonly needed tools, libraries etc – and then collaboratively building up those key components. Such a collaborative approach will depend on institutions, projects and individuals adopting new interdependencies and ways of working.

3.20 Performance portability: the good, the bad, and the ugly

Simon McIntosh-Smith (University of Bristol, GB)

URL http://uob-hpc.github.io

Achieving functional portability across a diverse range of computer architectures, such as CPUs and GPUs, is already a big challenge. Adding performance portability, where the same code performs well across those diverse architectures, is usually too ambitious a goal for scientific software developers. But what are the fundamental reasons behind this problem? Today, several parallel programming models can target a diverse range of hardware platforms: OpenMP 4.5, OpenCL, Kokkos and SYCL are just a small list of open source approaches for cross platform parallel programming. Why can't we write one program in, say OpenMP 4.5, and have this one code run well on GPUs from NVIDIA and AMD, and CPUs from Intel, IBM, Cavium et al. What are the fundamental technical problems that make this hard? And if we can enumerate and quantify these reasons, how can we then consciously design codes to avoid the main performance portability inhibitors?

In my HPC research group we have been studying performance portability since 2009. We have collected numerous case studies, where some codes are naturally performance portable, some can be adapted to become performance portable, while others appear to be naturally hostile to performance portability. Our initial research used OpenCL, but in the last few years our focus has been on OpenMP 4.x, and the emerging C++ parallel programming models that support cross-platform code generation (Kokkos, SYCL, Raja et al). Our target application areas have included life science codes (mostly molecular dynamics), and multi-physics codes (such as particle transport, heat diffusion and hydrodynamics).

In this talk I will describe our performance portability findings, including what we've found does work, what doesn't work, and where we think the most interesting open questions are.

3.21 AnyDSL: A Compiler-Framework for Domain-Specific Libraries (DSLs)

Richard Membarth (DFKI – Saarbrücken, DE)

AnyDSL is a framework for the rapid development of domain-specific libraries (DSLs). AnyDSL's main ingredient is AnyDSL's intermediate representation Thorin. In contrast to other intermediate representations, Thorin features certain abstractions which allow to maintain domain-specific types and control-flow. On these grounds, a DSL compiler gains two major advantages:

- The domain expert can focus on the semantics of the DSL. The DSL's code generator can leave low-level details like exact iteration order of looping constructs or detailed memory layout of data types open. Nevertheless, the code generator can emit Thorin code which acts as interchange format.
- The expert of a certain target machine just has to specify the required details once. These details are linked like a library to the abstract Thorin code. Thorin's analyses and transformations will then optimize the resulting Thorin code in a way such that the resulting Thorin code appears to be written by an expert of that target machine.

3.22 If you've scheduled loops, you've gone too far

Lawrence Mitchell (Imperial College London, GB)

Joint work of David A. Ham, Miklós Homolya, Paul H. J. Kelly, Fabio Luporini, Lawrence Mitchell
Main reference Miklós Homolya, Lawrence Mitchell, Fabio Luporini, David A. Ham: "TSFC: a structure-preserving form compiler", CoRR abs/1705.03667, 2017.
URL https://arxiv.org/abs/1705.003667

The optimal loop schedule for a given algorithm is typically hardware dependent, even with all other parameters of the algorithm fixed. When we manually port code to a new hardware platform, we must understand the loop structure and then perform, in tandem, data layout and loop reordering to achieve good performance. This is a difficult task. I argue that requiring a compiler system to perform the same task will never work: the scheduled loop nest does not offer enough information to the compiler for it to determine the algorithmic structure. Instead, we should strive for program transformation steps that operate on *unscheduled* DAGs. This is most easily achieved with DSLs, since no analysis is required. I will say some things about how we achieve this in the context of finite element codes, but will mostly be full of questions.

References

1 F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. McRae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, *Firedrake: automating the finite element method by composing abstractions*, ACM Transactions on Mathematical Software, 43 (2016), pp. 24:1–24:27, https://doi.org/10.1145/2998441, https://arxiv.org/abs/1501.01809.
2 M. Homolya, R. C. Kirby, and D. A. Ham, *Exposing and exploiting structure: optimal code generation for high-order finite element methods*, 2017, https://arxiv.org/abs/1711.02473.

3.23 Composing parallel codes while preserving portability of performance

Raymond Namyst (University of Bordeaux, FR)

Joint work of Raymond Namyst, Andra Hugo, Terry Cojean, Nathalie Furmento, Samuel Thibault, Pierre-André Wacrenier

Future exascale computers are expected to exhibit an unprecedented degree of parallelism together with a deeply hierarchical architecture, which only a few experts will be able to exploit efficiently if no significant progress is made in HPC software development. This situation is even more regrettable because there already exist many candidate applications with the potential to occupy the massive number of computing resources promised by exascale machines. Coupled applications for instance, which typically implement multi-physics and/or multi-scale simulations, exhibit high degrees of parallelism.

For such applications, designing a completely new "exascale programming model" can not be the answer, as rewriting coupled applications from scratch would require state-of-the-art skills in several domains (e.g. astrophysics and machine learning). Building these applications by reusing existing codes is thus the only reasonable way to go. The main challenge is thus about enabling a smooth transition to exascale that would leverage a significant base of existing codes and applications.

Unfortunately, reusing parallel codes in HPC applications is not commonplace at all. The main reason lies in current implementations of parallel libraries not being ready to run simultaneously over the same hardware resources, causing resource oversubscription, scheduling interferences, and other problems linked to unawareness of resource usage and compatibility of execution models. This problem – known as the parallel composability problem – significantly limits the way parallel codes can interact together.

This presentation covers the research topics raised by the challenge of "efficient parallel code reuse". In the light of our experience in the design of the StarPU task-based runtime system, we discuss how these challenges could be addressed by future runtime systems and programming environments.

References

1 C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures, Concurr. Comput. : Pract. Exper. 23 (2011) 187–198.
2 H. Pan, B. Hindman, K. Asanović, Composing parallel software efficiently with lithe, SIGPLAN Not. 45 (2010) 376–387.

3.24 Performance Portability through Device Specialization

Simon Pennycook (Intel – Santa Clara, US)

Joint work of Simon Pennycook, Jason Sewall, Alejandro Duran, Victor Lee
Main reference S. J. Pennycook, J. D. Sewall, V. W. Lee, "Implications of a Metric for Performance Portability", In Future Generation Computer Systems, ISSN 0167-739X, 2017.
URL https://doi.org/10.1016/j.future.2017.08.007

We discuss the utility of shared definitions and metrics for "performance portability", and the implications of one such metric on the design of highly performance portable software. Specifically, we demonstrate that maintaining multiple implementations of some limited subset of application functionality can significantly improve performance portability, thereby motivating improved mechanisms for managing function variants in software.

3.25 Loop descriptors and cross-loop techniques: portability, locality and more

Istvan Reguly (Pazmany Peter Catholic University – Budapest, HU)

Joint work of Istvan Reguly, Mike Giles, Gihan Mudalige

This talk will outline what additional information can be added to "parallel for" loops about data accesses and how it enables high-level optimisations and portability. Adding in a user contract that data is not modified outside of these loops, we then show how the execution of these loops can be delayed, which enables cross-loop analysis and optimisations, showing a few examples involving cache-blocking tiling and checkpointing. This talk aims to prompt discussion on the further possible uses of per-loop and cross-loop techniques, and the relaxation of domain-specific semantics.

3.26 Characterizing Data Movement Costs: Tools Needed!

P. (Saday) Sadayappan (Ohio State University – Columbus, US)

Data movement overheads dominate operation execution costs, both in terms of time (performance) as well as energy. But while the computational complexity of algorithms (in terms of number of elementary operations) is well understood, the data-movement complexity of algorithms is not. A significant complication for users to reason about the inherent data-movement complexity of computations is that unlike computational complexity, it is not additive under composition. Tools for characterizing data-movement complexity will be important in facilitating high performance, productivity, and performance-portability with high-level frameworks.

3.27 Performance Portability Through Code Generation – Experiences in the context of SaC

Sven-Bodo Scholz (Heriot-Watt University – Edinburgh, GB)

URL www.sac-home.org

This talk gives a bird's eye view on the performance portability experiences made in the context of compiling SaC into high-performance codes for a range of architectures. Particular emphasis is being put on the challenges met, the interplay between language design and those challenges, as well as a discussion on how these experiences compare to other approaches such as DSLs, staging or library based solutions.

3.28 Orthogonal Abstractions for Scheduling and Storage Mappings

Michelle Mills Strout (University of Arizona – Tucson, US)

With ever increasing amounts of HW parallelism, it is becoming more and more critical to reduce synchronization overhead and improve the data locality of computations. Reducing synchronization overhead can be done by creating many, decent-sized tasks that are only loosely synchronized. Improving data locality requires that the computation placed within a single task reuses data and that that data fits within the memory resources where the task is allocated. Reducing temporary storage is important as well. These activities require scheduling across loops and in some cases function calls and modifying how data values are mapped to storage locations.

Systems do this in a multitude of ways. For this discussion, some example approaches from projects such as OP2, Chapel, Kokkos, CnC, and Loop Chaining will be overviewed. What might an orthogonal abstraction stack for performance programming look like? How would such an abstraction stack interact with performance portability?

3.29 Mis-predicting performance

Nathan Tallent (Pacific Northwest National Lab. – Richland, US)

Main reference Ryan D. Friese, Nathan R. Tallent, Abhinav Vishnu, Darren J. Kerbyson, Adolfy Hoisie: "Generating Performance Models for Irregular Applications", in Proc. of the 2017 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2017, Orlando, FL, USA, May 29 - June 2, 2017, pp. 317–326, IEEE Computer Society, 2017.
URL http://dx.doi.org/10.1109/IPDPS.2017.61

Performance modeling gives insight into bottlenecks to performance portability. There are several approaches to performance modeling. Analytical models promise insight by representing program behavior using algebraic expressions that are a function of input parameters. However, they can be time-consuming to create and may miss important corner cases. Statistical and machine learning models can be more easily automated. However, they easily confuse cause and correlation; and are only as good as their training sets.

This talk focuses on some of the open questions in modeling performance. We use examples from recent work on modeling applications and workflows with irregular behavior, e.g, input-dependent solvers, irregular memory accesses, or unbiased branches. This irregular behavior often leads to long-latency events that are difficult to characterize when moving between different systems or programming models.

3.30 Performance Portability with Pragmas – Wishful Thinking or Real Opportunity?

Christian Terboven (RWTH Aachen, DE)

Since the evolution of OpenMP and OpenACC to de-facto standards in the HPC community, directive-based parallelization offers a higher level of abstraction than system- or device-level APIs. Both standards hide certain details from the programmer to focus on expressing parallelism at different levels. To answer the question if directives can offer an opportunity to enable performance portable programming, this contribution summarized and discussed recent developments within OpenMP.

OpenMP 4.5 was released in 2015 and introduced a feature called locks with hints. It enables the use of different lock types within a single program. Furthermore, it allows the specification of a lock hint that the runtime can exploit to determine the most efficient available implementation. For example the lock hint ‚speculative' may be realized by hardware-supported memory transactions, if supported by the target architecture.

OpenMP TR5 was released in 2016 and provided an early view of memory management support scheduled for inclusion in OpenMP 5.0. The addition of allocators as a concept in OpenMP enables the use of different kinds of memory. Instead of targeting a dedicated memory kind with a device-specific API, OpenMP TR5 allows to allocate from memory with a specific feature, for example memory with the highest available bandwidth can be selected. The concrete memory kind will be selected at runtime based on capabilities of the target architecture.

Both examples illustrate how the use of directives can hide hardware-level details and how the runtime system can apply optimizations under the hood. OpenMP and OpenACC differ

in being prescriptive or descriptive. In the prescriptive approach of OpenMP, the programmer explicitly instructs how parallelism is extracted from the application code and mapped to the system. OpenACC tends to require fewer specific details from the programmer and leaves more freedom and room for optimization to the compiler and runtime. The use of directives over low-level APIs can contribute to the goal of performance portable programming, but has to be complemented by other measures.

References

1 H. Bae, J. Cownie, M. Klemm and C. Terboven: A User-Guided Locking API for the OpenMP Application Programming Interface. IWOMP 2014: 173–186, 2014.
2 J. Sewall, J. Pennycook, A. Duran, C. Terboven, X. Tian and R. Narayanaswamy: Developments in Memory Management in OpenMP. IJHPCN, to appear.

3.31 Thread and Data Placement on Multicore Architectures

Didem Unat (Koc University – Istanbul, TR)

Joint work of Didem Unat, Pirah Noor Soomro, Muhammad Aditya Sasongko, Mohammad Laghari
Main reference Mohammad Laghari, Didem Unat: "Object Placement for High Bandwidth Memory Augmented with High Capacity Memory", in Proc. of the 29th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2017, Campinas, Brazil, October 17-20, 2017, pp. 129–136, IEEE Computer Society, 2017.
URL http://dx.doi.org/10.1109/SBAC-PAD.2017.24

Placement of parallel tasks and data in current multicore machines is one of the key aspects to achieve good performance. A thread placement policy is formulated by discovering machine's topology and analyzing application's behavior of sharing data among parallel task. This policy is used for binding application threads during its execution. Current binding options supported by the runtime systems do not analyze application's behavior and only provide basic mapping policies. To improve the performance programmer has to put an extra effort to understand core topology and memory hierarchy, then bind the tasks accordingly. This process is not portable; programmer has to repeat the analysis when the platform or application changes. We propose a thread mapping algorithm based on the communication pattern and machine topology. Our algorithm is fair in other words, it tries to pair threads that communicate most with each other and considers mutual preferences.

In addition to thread placement, in a memory subsystem where there are different types of memories, placement of data objects becomes a programming issue because the programmer has to decide which objects to place on which types of memory. We propose an object placement algorithm that places program objects to fast or slow memories by considering characteristics of each memory type. Our algorithm uses the reference counts and type of references (read or write) to make an initial placement of data. By placing objects according to our placement algorithm, we are able to achieve a speedup of up to 2x with 6 applications under various system configurations.

3.32 A Heterogeneous Talk: thoughts on portability, heterogeneous computing, and graphs

Ana Lucia Varbanescu (University of Amsterdam, NL)

Joint work of Stijn Heldens, Ana Lucia Varbanescu, Alexandru Iosup
URL https://github.com/stijnh/HyGraph

In this talk, we discuss performance portability from the perspective of programming models and heterogeneous computing. We further discuss the implications of irregular applications and data-dependent performance variability on such issues as performance portability and heterogeneous computing. Finally, we briefly introduce HyGraph, our runtime-based solution for heterogeneous graph processing. HyGraph is a novel graph-processing system for hybrid platforms which delivers performance by utilizing both the CPU and the GPU concurrently. Contrary to the state-of-the-art approach of statically partitioning the workload beforehand, HyGraph delivers performance by dynamic scheduling of jobs onto both the CPU and the GPU, thus providing automatic load balancing and superseding the need for the user to manually define a static workload distribution. Additionally, HyGraph minimizes the inter-process communication overhead by carefully overlapping computation and communication. Our results demonstrate that HyGraph can deliver system-level "efficiency portability" on different, large-scale systems.

3.33 Challenges with Different approaches for Performance Portability

Mohamed Wahib (AIST – Tokyo, JP)

The approaches for tackling the performance portability problem typically fall under one, or a mix, of the following: DSLs, compiler-based approaches, source-to-source translators, and libraries. Considering the diversity in pros and cons for each of those approaches, it is hard for the programmer to decide on which approach to follow. This is especially challenging when the programmer has to address performance portability for legacy code. This presentation compares experiences in addressing performance portability, using each of the approaches mentioned above. Main pitfalls and limitations, for some legacy real-world applications, are also discussed.

3.34 Deep memory hierarchies and performance portability

Michele Weiland (University of Edinburgh, GB)

Memory hierarchies are becoming deeper and more complex, and byte-addressable SCM is blurring the line between memory and storage. How users and applications can benefit from new memory technologies depends on how new layers in the memory hierarchy are exposed and used. The question is: whose responsibility is it to guarantee transparent use of deep

memory hierarchies? Can the application developer expect this to be dealt with on a system level, or will it require direct intervention at the programming level? In this presentation I will look at the implications of complex memory hierarchies for performance and performance portability.

3.35 Musings on Performance Portability

Michael Wolfe (NVIDIA Corp., US)

The term "performance portability" has been used for at least twenty years. Some people argue that it is not achievable, and even that it's not desireable. This is folly. However, there is a tension between three P's: Performance, Productivity, and Portability. Each user or organization needs to optimize this function. Generally, any program can be made to run faster with greater programming effort, such as using machine language, but the productivity cost could be prohibitive. Similarly, programming in machine language reduces portability, even across different generations of the same instruction set architecture. But different people and organizations will have different pain thresholds for how much performance they are willing to give up to get some degree of productivity and of portability. Perhaps we should rename this discussion "Productive Performance Portability."

Let's review some successes in performance portability across the ages: 1957: The Fortran compiler from IBM, which had a goal to generate code that was as fast as hand-written machine language, allowed programs that could port across machines, and has been extremely successful for over 60 years now. 1977: Vectorizing compilers virtualized the details of vector code generation and allowed programmers to take advantage of vector instructions, and now SIMD instructions, without having to know details such as the vector or SIMD register length, and became the dominant method for programming vector computers. 1997: MPI was introduced earlier in the decade, but by 1977 it had replaced all previous message passing libraries, and remains the most common method for scalable programming, across thousands of nodes on a supercomputer network.

Now it's 2017, and we have highly scalable nodes, with many dozens or hundreds of compute units, all with shared memory or high-speed interconnected memories. Can we achieve productive performance portability across the range of such systems? I argue that we should be able to, because they all share many characteristics: Many processing elements, SIMD execution, multithreading, hardware caches. There have been a number of approaches, including languages, compilers, runtime systems, and numerical libraries. All of these may have a role to play. However, if there's a lesson to learn from history it's that we need to train programmers, many or most of whom are more interested in science than in programming, how to write programs that perform well and that will port to future computer systems as well. Vectorizing compilers played a role in training programmers for those machines, by giving immediate and precise feedback about how well the loops vectorized and why they did not. Such tools should be explored for future productive performance portability as well.

4 Open problems

4.1 Performance Portability via Automatic Region-based Auto-tuning

Philipp Gschwandtner (Universität Innsbruck, AT)

Joint work of Philipp Gschwandtner, Juan J. Durillo, Thomas Fahringer
Main reference Klaus Kofler, Juan José Durillo, Philipp Gschwandtner, Thomas Fahringer: "A Region-Aware Multi-Objective Auto-Tuner for Parallel Programs", in Proc. of the 46th International Conference on Parallel Processing Workshops, ICPP Workshops 2017, Bristol, United Kingdom, August 14-17, 2017, pp. 190–199, IEEE Computer Society, 2017.
URL http://dx.doi.org/10.1109/ICPPW.2017.37

Optimizing parallel programs for modern architectures and striving for performance portability is a notoriously difficult task. The increasing complexity of multi- and many-core platforms, hardware details such as topologies, caches, and links and multiple layers of nested parallelism pose a limiting factor when designing both faster hardware and software in contemporary HPC. Auto-tuning has become increasingly popular to mitigate this issue, but still lacks key features for pervasive use throughout the community. First, many auto-tuner systems require user directives that e.g. describe performance relationships or bottlenecks, or specify a list of tunable parameters and ranges of valid settings. Second, most parallel programs can be subdivided into several regions, whose optimal tunable parameter settings might differ, and whose optimization might have adverse effects on subsequent regions - harming overall performance. These issues are greatly aggravated by parameter setting overheads, non-contiguous, combinatorial parameter spaces, and multi-objective optimization environments. To address this problem, a fully automatic, region-based auto-tuner is needed, minimizing user effort in realizing performance portability. The auto-tuner should automatically identify program code regions of interest, find promising parameter spaces for a given set of user objectives, and explore them efficiently.

Participants

- Sadaf Alam
CSCS – Lugano, CH
- Michael Bader
TU München, DE
- Carlo Bertolli
IBM TJ Watson Research Center – Yorktown Heights, US
- Mauro Bianco
CSCS – Lugano, CH
- Alexandru Calotoiu
TU Darmstadt, DE
- Bradford Chamberlain
Cray Inc. – Seattle, US
- Aparna Chandramowlishwaran
University of California – Irvine, US
- Kemal A. Delic
Hewlett Packard – Grenoble, FR
- Christophe Dubach
University of Edinburgh, GB
- Anshu Dubey
Argonne National Laboratory, US
- H. Carter Edwards
Sandia National Labs – Albuquerque, US
- Jan Eitzinger
Universität Erlangen-Nürnberg, DE
- Todd Gamblin
LLNL – Livermore, US
- Lin Gan
Tsinghua University – Beijing, CN
- William D. Gropp
University of Illinois – Urbana-Champaign, US
- Philipp Gschwandtner
Universität Innsbruck, AT
- Mary W. Hall
University of Utah – Salt Lake City, US
- Robert J. Harrison
Brookhaven National Laboratory – Upton, US
- Alexandra Jimborean
Uppsala University, SE
- Paul H. J. Kelly
Imperial College London, GB
- Andreas Klöckner
University of Illinois – Urbana-Champaign, US
- Kathleen Knobe
Rice University – Houston, US
- Seyong Lee
Oak Ridge National Laboratory, US
- Naoya Maruyama
LLNL – Livermore, US
- Chris Maynard
MetOffice – Exeter, GB
- Simon McIntosh-Smith
University of Bristol, GB
- Richard Membarth
DFKI – Saarbrücken, DE
- Lawrence Mitchell
Imperial College London, GB
- Bernd Mohr
Jülich Supercomputing Centre, DE
- Raymond Namyst
University of Bordeaux, FR
- Simon Pennycook
Intel – Santa Clara, US
- Istvan Reguly
Pazmany Peter Catholic University – Budapest, HU
- P. (Saday) Sadayappan
Ohio State University – Columbus, US
- Sven-Bodo Scholz
Heriot-Watt University – Edinburgh, GB
- Michelle Mills Strout
University of Arizona – Tucson, US
- Nathan Tallent
Pacific Northwest National Lab. – Richland, US
- Christian Terboven
RWTH Aachen, DE
- Didem Unat
Koc University – Istanbul, TR
- Ana Lucia Varbanescu
University of Amsterdam, NL
- Jeffrey S. Vetter
Oak Ridge National Laboratory, US
- Mohamed Wahib
AIST – Tokyo, JP
- Michele Weiland
University of Edinburgh, GB
- Robert Wisniewski
Intel – Santa Clara, US
- Michael Wolfe
NVIDIA Corp., US