PIPER: Performance Insight for Programmers and Exascale Runtimes

Guiding the Development of the Exascale Software Stack

Final Report

Rice University Subproject

Cooperative Agreement No. DE-SC0010473

September 2013–August 2016

John Mellor-Crummey

Principal Investigator at Rice University

Department of Computer Science, MS 132
Rice University
P.O. Box 1892
Houston, TX 77251-1892
Voice: 713-348-5179
FAX: 713-348-5930
Email: johnmc@rice.edu

Contents

1 Introduction **1**

2 Summary Description of the Project's Research and Development **1**

2.1 New Measurement Capabilities . . . 1

2.1.1 Data-centric Profiling of Parallel Programs . . . 1

2.1.2 Sampling-based Performance Analysis of GPU-accelerated Supercomputers . 1

2.1.3 Pinpointing Data Locality Bottlenecks with Low Overhead . . . 2

2.1.4 Performance Measurement and Analysis of OpenMP Programs . . . 2

2.1.5 Measurement and Analysis of Threaded Programs on NUMA Architectures . 2

2.1.6 Lightweight Profiling to Guide Array Regrouping . . . 3

2.1.7 Blame Shifting for Root Cause Analysis . . . 3

2.2 Runtime Interfaces for Tools . . . 3

2.2.1 OMPT: A Performance Tools Interface for OpenMP . . . 3

2.2.2 OMPD: A Debugging Interface for OpenMP . . . 4

2.3 New Attribution Capabilities . . . 5

2.3.1 Binary Analysis to Improve Performance Attribution to Optimized C++ Codes 5

2.3.2 Efficient Attribution using Fine-grained Binary Instrumentation . . . 5

2.4 New Presentation Capabilities . . . 6

2.4.1 Scaling Visualization of Performance Data . . . 6

2.5 Application Engagement Outcomes . . . 6

2.5.1 Analysis and Optimization of Applications . . . 6

2.5.2 Analysis and Optimization of a Communication Library . . . 7

2.5.3 Efficient Synchronization for Parallel Systems . . . 7

2.5.4 Contention-conscious, Locality-preserving Locks . . . 7

2.6 Interactions with other X-Stack Projects . . . 8

3 Research Objectives Remaining **8**

4 Findings **9**

4.1 Hardware Technology. . . . 9

4.2 Software Technology . . . 10

5 Products of the Research **11**

5.1 Contributions to Community Standards . . . 11

5.1.1 OMPT Tools API for OpenMP . . . 11

5.1.2 OMPD Debugging API for OpenMP . . . 11

5.2 Open Source Software . . . 11

5.2.1 HPCToolkit . . . 11

5.2.2 OMPT Tools API in LLVM OpenMP . . . 12

5.2.3 OMPT Test Suite for OpenMP . . . 12

5.3 Technical Communications . . . 12

5.3.1 Presentations . . . 12

5.3.2 Publications . . . 13

5.3.3 Reports . . . 15

5.3.4 Videos . . . 15

1 Introduction

The aim of the PIPER project was to develop methodologies and software for measurement, analysis, attribution, and presentation of performance data for extreme-scale systems. Goals of the project were to support analysis of massive multi-scale parallelism, heterogeneous architectures, multi-faceted performance concerns, and to support both post-mortem performance analysis to identify program features that contribute to problematic performance and on-line performance analysis to drive adaptation.

Producing a complete suite of performance tools for exascale platforms during the course of this project was impossible since both hardware and software for exascale systems is still a moving target. For that reason, the project focused broadly on the development of new techniques for measurement and analysis of performance on modern parallel architectures, enhancements to HPCToolkit's software infrastructure to support our research goals or use on sophisticated applications, engaging developers of multithreaded runtimes to explore how support for tools should be integrated into their designs, engaging operating system developers with feature requests for enhanced monitoring support, engaging vendors with requests that they add hardware measurement capabilities and software interfaces needed by tools as they design new components of HPC platforms including processors, accelerators and networks, and finally collaborations with partners interested in using HPCToolkit to analyze and tune scalable parallel applications.

Following the investments in HPCToolkit as part of the PIPER project, HPCToolkit was selected as a key performance tools technology for the DOE's Exascale Computing Project.

2 Summary Description of the Project's Research and Development

2.1 New Measurement Capabilities

2.1.1 Data-centric Profiling of Parallel Programs

It is difficult to manually identify opportunities for enhancing data locality. To address this problem, we extended the HPCToolkit performance tools to support data-centric profiling of scalable parallel programs. HPCToolkit uses hardware counters to directly measure memory access latency and attributes latency metrics to both variables and instructions. Different hardware counters provide insight into different aspects of data locality (or lack thereof). Unlike prior tools for data-centric analysis, HPCToolkit employs scalable measurement, analysis, and presentation methods that enable it to analyze the memory access behavior of scalable parallel programs with low runtime and space overhead. We demonstrated the utility of HPCToolkit's new data-centric analysis capabilities with case studies of five well-known benchmarks. In each benchmark, we identify performance bottlenecks caused by poor data locality and demonstrate non-trivial performance optimizations enabled by this guidance.

2.1.2 Sampling-based Performance Analysis of GPU-accelerated Supercomputers

Performance analysis of GPU-accelerated systems requires a system-wide view that considers both CPU and GPU components. We extended HPCToolkit to support system-wide, sampling-based performance analysis methods of GPU-accelerated systems. Since current GPUs do not support sampling, our implementation required careful coordination of instrumentation-based performance data collection on GPUs with sampling-based methods employed on CPUs. In addition, we developed a novel technique for analyzing systemic idleness in CPU/GPU systems. We demonstrated the effectiveness of our techniques with application case studies on Titan and Keeneland. Some highlights of our case studies include improving performance for LULESH 1.0 by 30%, identifying a hardware performance problem on Keeneland, identifying a scaling problem in LAMMPS derived from CUDA initialization, and identifying a performance problem that is caused by GPU synchronization operations that suffer delays due to blocking system calls.

2.1.3 Pinpointing Data Locality Bottlenecks with Low Overhead

A wide gap exists between the speed of modern processors and memory subsystems. As a result, long latencies associated with fetching data from memory often significantly degrade execution performance. To aid with program tuning, application developers need tools to analyze memory access patterns and guide them how to reuse data in the fastest levels of a system's memory hierarchy. To address this problem, we developed a novel, efficient, and effective tool for data locality measurement and analysis. Unlike other tools, our tool uses both statistical PMU sampling to quantify the cost of data locality bottlenecks and cache simulation to compute reuse distance to diagnose the causes of locality problems. This approach enables us to collect rich information to provide insight into a program's data locality problems. Our tool attributes quantitative measurements of observed memory latency to program variables and dynamically allocated data, as well as code. Our tool identifies data touched by reuse pairs and the accesses involved, identified with their full calling context. Finally, our tool employs both sampling and parallelization to accelerate the computation of representative reuse distance information. Experiments show that with an overhead of only about 13%, our tool provides detailed insights that enabled us to make non-trivial improvements to memory-bound HPC benchmarks.

2.1.4 Performance Measurement and Analysis of OpenMP Programs

The number of hardware threads is growing with each new generation of multicore chips; thus, one must effectively use threads to fully exploit emerging processors. OpenMP is a popular directive-based programming model that helps programmers exploit thread-level parallelism. To support measurement and analysis of OpenMP, we extended HPCToolkit in two principal ways. First, we developed a measurement methodology that attributes blame for work and inefficiency back to program contexts. We showed how to integrate prior work on measurement methodologies that employ directed and undirected blame shifting and extend the approach to support dynamic thread-level parallelism in both time-shared and dedicated environments. Second, we developed a novel deferred context resolution method that supports online attribution of performance metrics to full calling contexts within an OpenMP program execution. This approach enables us to collect compact call path profiles for OpenMP program executions without the need for traces. Support for our approach motivated much of the design for a new performance tools interface for OpenMP known as OMPT. We demonstrated the effectiveness of our approach by applying our tool to analyze four well-known application benchmarks that cover the spectrum of OpenMP features. In case studies with these benchmarks, insights from our tool helped us significantly improve the performance of these codes.

2.1.5 Measurement and Analysis of Threaded Programs on NUMA Architectures

Almost all of today's microprocessors contain memory controllers and directly attach to memory. Modern multiprocessor systems support non-uniform memory access (NUMA): it is faster for a microprocessor to access memory that is directly attached than it is to access memory attached to another processor. Without careful distribution of computation and data, a multithreaded program running on such a system may have high average memory access latency. To use multiprocessor systems efficiently, programmers need performance tools to guide the design of NUMA-aware codes. To address this need, we enhanced the HPCToolkit performance tools to support measurement and analysis of performance problems on multiprocessor systems with multiple NUMA domains. With these extensions, HPCToolkit helps pinpoint, quantify, and analyze NUMA bottlenecks in executions of multithreaded programs. It computes derived metrics to assess the severity of bottlenecks, analyzes memory accesses, and provides a wealth of information to guide NUMA optimization, including information about how to distribute data to reduce access latency and minimize contention.

We demonstrate the utility of our approach through case studies in which we use these capabilities to diagnose NUMA bottlenecks in four multithreaded applications.

2.1.6 Lightweight Profiling to Guide Array Regrouping

Memory hierarchies in modern computer systems are complex; often, they include multi-level caches and multiple memory controllers on the same chip. Without careful design, programs suffer from unnecessary data movement between caches and memory, degrading performance and increasing energy consumption. Array regrouping can significantly improve data locality by improving spatial reuse of data and reducing cache contention. However, existing techniques for identifying opportunities for array regrouping are lacking in three ways. First, they provide inadequate information to guide regrouping. Second, the cost of monitoring employed by prior tools to identify regrouping opportunities limits the use of these methods in practice. Third, existing metrics for quantifying the benefits of array regrouping can lead to inappropriate transformations that hurt performance. To address these issues, we developed ArrayTool - a lightweight profiler that guides array regrouping. ArrayTool has three unique capabilities. First, it focuses attention on arrays with significant access latency. Second, it identifies the feasibility and quantifies the benefits of regrouping arrays with lightweight array-centric profiling. Third, it works on both shared-memory and distributed-memory parallel programs. To illustrate the utility of ArrayTool, we employ it to analyze three benchmarks. Using the guidance it provides, we regroup program arrays, improving performance from 25% to a factor of two.

2.1.7 Blame Shifting for Root Cause Analysis

Blame shifting [10, 11] is a performance measurement and analysis strategy that identifies root causes for idleness by shifting blame for thread idleness to threads causing it by holding a lock or executing code that lacks sufficient parallelism to keep all threads busy. As part of PIPER, we worked to extend and generalize the blame-shifting approach for arbitrary multithreaded models. In our prior work, we explored *directed* blame shifting to attribute lock waiting to lock holders in the Pthreads programming model, and *undirected* blame shifting to attribute waiting for work to insufficiently parallel code in the context of Cilk programs. In this project, we extended and integrated these ideas to analyze both kinds of waiting in executions of node programs that use OpenMP. Our insights from this work were incorporated into the design of features to support blame shifting in the OMPT performance tools API for OpenMP. In addition to our work with OpenMP, we developed a generalization of blame shifting for pinpointing bottlenecks associated with the use of spin waiting for locks or blocking for locks or condition variables in programs using the library-based Pthreads threading abstractions. Furthermore, we explored generalizing blame shifting to parallel programming models at multiple levels of abstraction. To support this, we developed a prototype of a *modular* blame-shifting framework. Using this framework, blame-shifting at each level of abstraction is implemented independently of all others. Results with this framework were promising and we believe it could be employed to monitor complex programming abstractions such NWChem's use of Global Arrays in conjunction with Intel's Threaded Building Blocks atop Pthreads.

2.2 Runtime Interfaces for Tools

2.2.1 OMPT: A Performance Tools Interface for OpenMP

A goal of the PIPER project was to analyze the requirements for tool APIs at various levels in the exascale software stack. As a first step toward understanding the needs for tool APIs for multithreaded runtime systems in general, Martin Schulz (LLNL) and John Mellor-Crummey (Rice University) of the PIPER project and Alexandre Eichenberger (IBM) led the design of OMPT—an interface for monitoring and analysis tools to be supported by the OpenMP runtime. OpenMP was chosen as the target for this work as it is the most widely used multithreaded programming model for scientific computing. Development of the OMPT interface was a community effort with broad participation by members of the OpenMP Tools Subcommittee.

Initially, we designed the OMPT API to support monitoring of OpenMP 3.1 features, focusing on parallel regions, worksharing constructs, and tasking. Subsequently, we extended OMPT to include support for monitoring of data movement between CPUs and accelerators as well as an interface for collecting and inspecting detailed traces of activity on accelerators. The final design for the OMPT API provides a mechanism to initialize a first-party tool, routines that enable a tool to determine the capabilities of an OpenMP implementation, routines that enable a tool to examine OpenMP state information associated with a thread, mechanisms that enable a tool to map implementation-level calling contexts back to their source-level representations, a callback interface that enables a tool to receive notification of OpenMP events, a tracing interface that enables a tool to trace activity on OpenMP target devices, and a runtime library routine that an application can use to control a tool.

To promote acceptance of the OMPT performance tools API, the PIPER team at Rice led the development and validation of an implementation of an early design of the OMPT API in the context of Intel's open source OpenMP runtime. In May 2015, after our implementation of OMPT became sufficiently mature, it was merged into the trunk of the LLVM OpenMP runtime system. The LLVM OpenMP runtime and ecosystem is a key part of the software environment for the forthcoming DOE CORAL platforms.

To validate implementations of OMPT, the PIPER team at Rice developed a test suite that exercises an OpenMP runtime by driving it through certain known states, and uses OMPT to monitor the test programs to validate that OMPT callbacks functions and inquiry functions return the proper information to describe these known states. Our prototype test suite was subsequently refined by researchers at RWTH Aachen University and contributed to the LLVM OpenMP test suite to validate its OMPT implementation.

Shortly after the end of the PIPER project, OMPT was approved for inclusion into OpenMP 5.0. Ongoing work is focused on polishing the OMPT design in preparation for the offical release of the OpenMP 5.0 standard in 2018. Work to bring the implementation of OMPT in LLVM OpenMP into compliance with the emerging OpenMP 5.0 specification is ongoing.

2.2.2 OMPD: A Debugging Interface for OpenMP

Martin Schulz (LLNL) and John Mellor-Crummey (Rice University) of the PIPER project participated in the development of OMPD—a debugging API for OpenMP. The goals for the OMPD API were that it should enable a debugger to inspect the state of a live process or a core file, facilitate interactive control of a live process (including the ability to place breakpoints at the beginning and end of parallel regions and tasks), and shouldn't place an unreasonable burden on either runtime or tool developers.

The design for OMPD employs an approach previously employed to support debugging of threads and MPI programming abstractions: the programming abstraction provides a plugin library that the debugger loads into its own address space. The debugger then uses an API provided by the plugin library to inspect and manipulate state associated with the programming abstraction in a target. The target may be a live process or a core file.

Based on the initial design of OMPD, groups at LLNL and Rogue Wave constructed prototype implementations. Following successful prototyping, work to integrate OMPD into the OpenMP 5.0 standard has begun.

2.3 New Attribution Capabilities

2.3.1 Binary Analysis to Improve Performance Attribution to Optimized C++ Codes

A key component of HPCToolkit is hpcstruct—a utility that analyzes a program binary to recover and integrate information about functions, loops, inlined call chains, and source lines. Program structure information is used by HPCToolkit to map performance measurements back to application source code and provide the framework for what is ultimately displayed in the HPCToolkit's hpcviewer graphical user interface, which displays a source-code oriented view.

In the course of the PIPER project, we overhauled hpcstruct to use the SymtabAPI and ParseAPI components from Dyninst—a binary analysis toolkit developed by the PIPER team at Wisconsin. Following our renovations, hpcstruct now uses ParseAPI to reconstruct function control-flow graphs (CFG) from a program's machine code and SymtabAPI to read line maps and information about inlining from application binaries. Dyninst ParseAPI provides sophisticated analysis, deftly handling jump tables, tail calls and non-returning functions. The precision of Dyninst's analysis improved HPCToolkit's attribution and display of performance measurements.

A key technical challenge addressed by our overhaul of hpcstruct was combining information about loop nests with information about inlined call chains. Given a sequence of instructions along with information about the inlined call chain associated with each instruction and the loops present, the key challenge is determining whether the information indicates an inlined call chain that contains a loop, a loop whose body contains inlined functions, or some combination of the two. To address this challenge, we constructed a new data structure called the *inline tree* to represent the placement of statements and loops in the context of inline sequences. The inline tree enables a topological placement of loops in the proper context of inlined instructions. This approach also enables more accurate location of loop headers.

A major benefit of the overhaul of hpcstruct is better attribution of templated C++ code. Modern programming models such as the DEGAS HC++, Livermore's RAJA and Sandia's Kokkos rely on heavily templated code. Using such programming models, generated code is often far removed from an application's original source. Expanding C++ template abstractions often involves code from multiple files along with deep call chains of inlined code that contain loops at one or more levels. Using information hpcstruct recovers about inlined call chains using Dyninst's SymtabAPI, hpcviewer can display all the steps in the sequence from the original application code, through multiple files and templates down to the final line that generated the instruction.

2.3.2 Efficient Attribution using Fine-grained Binary Instrumentation

Fine-grained binary instrumentation is a popular technique to monitor program execution. Intel's Pin is a leading dynamic binary instrumentation framework for building program measurement and analysis tools. A key feature missing in Pin is the ability to associate call paths with instructions as they execute. The availability of calling context information enables Pin tools to provide more detailed diagnostic feedback. To address this challenge and support the needs of PIPER collaborators at PNNL, the Rice PIPER team developed CCTLib—a call path collection library that any Pin tool can use to obtain the full calling context at any and every machine instruction that executes. CCTLib not only associates any instruction with source code along the call path, but also points to the data object accessed by the instruction if it is a memory access.

With CCTLib, collecting call paths on each executed instruction is possible, even for reasonably long running programs. Prior art in call path collection for Pin has several limitations. Compared to other open-source Pin tools for call path collection, CCTLib provides richer information that is accurate even for programs with complex control flow and does so with about 30% less overhead—a difference of 14.

2.4 New Presentation Capabilities

2.4.1 Scaling Visualization of Performance Data

To support large-scale visualization of huge execution traces on a tiled display wall, we employ a client-server approach for examining execution traces: the server gathers and filters execution-trace data on the supercomputer, while the client presents it. This approach enables visualization of huge traces without the need to move them from the file system where they were recorded. The server is itself a parallel program written using MPI. The server quickly extracts a subset of a huge execution trace in response to requests from `hpctraceviewer`'s visualization client.

To accelerate rendering of large-scale visualizations, we also overhauled `hpctraceviewer`'s client-side rendering engine. The visualization client is built using the Eclipse Standard Windowing Toolkit (SWT). On Linux, SWT is built atop GTK—a multi-platform toolkit for creating graphical user interfaces. Although GTK is thread-aware, it is not thread-safe: each GTK operation holds a global lock as it executes. This is an obstacle if one wants to accelerate rendering by using multiple threads. To minimize serialization due to the global lock, we refactored `hpctraceviewer`'s visualization client to employ multiple threads to turn execution trace data into sequences of primitive rendering operations that they pass to a single rendering thread using a concurrent queue. Using this client-server approach, we were able to visualize large traces on a Rice University visualization wall with 7680 x 4320 resolution in under 10 seconds. Before the refactoring, such visualizations took several minutes to render, which hindered interactive exploration of execution traces.

2.5 Application Engagement Outcomes

As part of our work on PIPER, we collaborated with application and library developers in industry, academia, national laboratories, and ASCR co-design centers to understand the nature of challenges faced by developers of different kinds of applications and to motivate development of features in PIPER tools. In the course of this work, we worked with them to use HPCToolkit to gain insight into a range of scientific and commercial applications as well as communication libaries. Below, we describe a few results of our application engagement efforts.

2.5.1 Analysis and Optimization of Applications

Here, we highlight the results of application engagements with researchers at LBNL regarding the NWChem quantum chemistry code and with Shell Research regarding a parallel reverse time migration code. Insights to HPCToolkit provided insights that drove the application optimization work described below, which was supported by supplemental funding from LBNL and Shell.

Barrier Elision for Production Parallel Programs. NWChem is a large scientific code that employs multiple layers of programming abstractions. Within each programming abstraction, programmers typically employ conservative synchronization patterns, which leads to suboptimal performance when abstractions are layered. To address this problem in general beyond NWChem, we developed context-sensitive dynamic optimizations that elide barriers that are redundant during the program execution. Our approach uses on-the-fly data race detection to identify redundant barriers in their calling contexts; after an initial learning phase, our framework starts eliding all future barrier instances that occur in the same calling context. We applied our techniques to NWChem—a 6 million line computational chemistry code written in C/C++/Fortran that uses multiple layers of runtime libraries including the Global Array toolkit, ComEx, DMAPP, and MPI. Our technique elided a surprisingly high fraction of barriers (as many as 63%) in production runs. This redundancy elimination translates to application speedups as high as 14% on 2048 cores. Our techniques also provided valuable insight about the application behavior, later used by NWChem developers. Overall, we demonstrate the value of holistic context-sensitive analyses that consider the domain science in conjunction with the associated runtime software stack.

Analysis and Optimization of Distributed Reverse Time Migration Code. Applications to process seismic data employ scalable parallel systems to produce timely results. To fully exploit emerging processor architectures, applications need to employ threaded parallelism within nodes and message passing across nodes. Today, MPI+OpenMP is the preferred programming model for this task. However, tuning hybrid programs for clusters is difficult. Performance tools can help users identify bottlenecks and uncover opportunities for improvement. We used HPCToolkit to gain insight into the performance of a Shell MPI+OpenMP code that performs Reverse Time Migration (RTM) on a cluster of multicore processors. HPCToolkit provided us with insight into the effectiveness of the application's domain decomposition strategy, its use of threaded parallelism, and its utilization of functional units in the cores. By applying insights obtained from HPCToolkit, we were able to improve the performance of the Shell RTM code by roughly 30%.

2.5.2 Analysis and Optimization of a Communication Library

Data movement in high-performance computing systems accelerated by graphics processing units (GPUs) remains a challenging problem. Data communication in popular parallel programming models, such as the Message Passing Interface (MPI), is currently limited to the data stored in the CPU memory space. Auxiliary memory systems, such as GPU memory, are not integrated into such data movement standards, thus providing applications with no direct mechanism to perform end-to-end data movement. MPI-ACC, an integrated and extensible framework that allows end-to-end data movement in accelerator-based systems, was developed to address this problem. In a partnership with the MPI-ACC team, we employed HPCToolkit's capabilities for measuring performance on GPU-accelerated platforms to evaluate and tune the performance of MPI-ACC.

2.5.3 Efficient Synchronization for Parallel Systems

In the course of our work with applications, we found that efficient synchronization was a pervasive problem. As a result, as part of our application engagement efforts, we explored new algorithms for mutual exclusion to accelerate applications sharing mutable data structures.

High performance locks for multi-level NUMA systems Efficient locking mechanisms are critically important for high performance computers. On highly-threaded systems with a deep memory hierarchy, the throughput of traditional queueing locks, e.g., MCS locks, falls off due to NUMA effects. Two-level cohort locks perform better on NUMA systems, but fail to deliver top performance for deep NUMA hierarchies. To address this problem, we developed a hierarchical variant of the MCS lock that adapts the principles of cohort locking for architectures with deep NUMA hierarchies. Analytical models for throughput and fairness of Cohort-MCS (C-MCS) and Hierarchical MCS (HMCS) locks enable us to tailor these locks for high performance on any target platform without empirical tuning. Using these models, one can select parameters such that an HMCS lock will deliver better fairness than a C-MCS lock for a given throughput, or deliver better throughput for a given fairness. Our experiments show that, under high contention, a three-level HMCS lock delivers up to 7.6x higher lock throughput than a C-MCS lock on a 128-thread IBM Power 755 and a five-level HMCS lock delivers up to 72x higher lock throughput on a 4096-thread SGI UV 1000. On a K-means clustering code from MineBench, a three-level HMCS lock reduces its running time by up to 55% compared to the C-MCS lock on an IBM Power 755.

2.5.4 Contention-conscious, Locality-preserving Locks

NUMA-aware locks, such as the HMCS lock described above, exploit locality of reference among nearby threads to deliver high lock throughput under high contention. However, the hierarchical nature of such locality-aware locks increases latency, which reduces the throughput of uncontended or lightly-contended critical sections. Before this work, no lock design for NUMA systems has delivered both low latency under low contention and high throughput under high contention.

To address this issue, we developed an adaptive mutual exclusion scheme (AHMCS lock), which employs several orthogonal strategies—a hierarchical MCS (HMCS) lock for high throughput under high contention, Lamport's fast path approach for low latency under low contention, an adaptation mechanism that employs hysteresis to balance latency and throughput under moderate contention, and hardware transactional memory for lowest latency in the absence of contention. The result is a top performing lock that has most properties of an ideal mutual exclusion algorithm. AHMCS exploits the strengths of multiple contention management techniques to deliver high performance over a broad range of contention levels. Our empirical evaluations demonstrate the effectiveness of AHMCS over prior art.

2.6 Interactions with other X-Stack Projects

DEGAS The PIPER team at Rice University worked closely with with the DEGAS project to assess the performance characteristics of applications that employ dynamic global address space programming models. An effort in this area of particular note was a collaboration on performance analysis and optimization of a global-address space implementation of NWChem.

Traileka Glacier. HPCToolkit has been used by members of the X-Stack Traileka Glacier project's OCR team to assess the performance of codes both within and across the nodes of parallel systems. In collaboration with members of the OCR team, we sketched a design of the mechanisms necessary to extend the blame shifting approach for performance monitoring to the data-driven futures in OCR. Members of the OCR team developed interfaces to OCR to increase the benefits of sampling-based performance analysis with HPCToolkit.

EXPRESS. The PIPER team met with HPX runtime developers of the X-Stack XPRESS project to assess tool capabilities that needed to diagnose performance problems in codes implemented with HPX. In discussions with the HPX team, we determined that there were no obstacles to augmenting HPX with a few necessary callbacks needed for HPCToolkit to employ blame shifting for performance analysis of multithreaded computations. A shortage of resources in the XPRESS projects for collaboration in this area caused this effort to be abandoned.

3 Research Objectives Remaining

Hardware and software for exascale systems is still a moving target. The diverse architectural paths of today's systems posed a wide target for tools, leading us to explore not only tools and runtime support for both manycore CPUs and accelerators. Our research developing runtime and tool support for CPUs and GPUs led us to conclude that GPUs require fundamentally different support from CPUs. For that reason, our efforts developing runtime and tools support for CPU and GPU required code bases that are largely separate. As a result, coping with both CPU and GPU platforms consumed more project time and effort than expected.

Beyond our work on performance tools themselves, a substantial fraction of project effort was devoted to exploration of tool interfaces needed elsewhere in the software stack. A particular focus of our effort was on tool interfaces for the OpenMP runtime, which supports both CPU and GPU computing with a range of programming styles. The complexity of the design, implementation, and validation of the OMPT performance tools API for OpenMP surprised us and consumed a substantial fraction of project effort. While this effort paid off with OMPT being accepted for inclusion into the OpenMP 5.0 standard, it consumed time and effort that we had expected to spend on other aspects of the project.

One unmet objective for the Rice work on PIPER includes work with adaptive applications to employ on-line performance analysis for application steering. While it is believed that adaptive approaches will be important for reducing power consumption at exascale, we didn't encounter any applications of this sort in our work. Without adaptive algorithms to study, we lacked examples to motivate the development of techniques to measure the effectiveness of adaptive algorithms.

A second unmet objective includes depositing analysis results of HPCToolkit into a scalable data store that includes information from disparate data sources. During the course of the PIPER project, project resources were focused more on the design and implementation of measurement infrastructure to collect information from hardware and runtime systems and less on supporting more general post-mortem analysis of measurement data. An emerging collaboration with LLNL aims to address this issue.

4 Findings

4.1 Hardware Technology.

Exascale hardware will require integrated hardware support for performance measurement. Previously, monitoring support has been designed by architects without much input from tool developers about measurement needs. This needs to change. There are several areas in which hardware monitoring for exascale systems will need improvement.

- *Monitoring CPU performance.* Understanding application performance on modern processors with out-of-order cores is quite difficult because of the complexity of their microarchitectures. Furthermore, microarchitectures in today's parallel systems vary widely, ranging from the latency-optimized cores in IBM Power and Intel Xeon multicore processors to the throughput-optimized cores in Intel's Xeon Phi manycore processors. While today's architectures have a wealth of hardware performance counters that can provide deep insight into performance issues, a detailed methodology about how to use them is often lacking. While vendors have made some progress on providing hardware counters and top-down methodologies for using them to identify performance bottlenecks, e.g., [8, 13, 6], at this writing, a top-down methodology is not available for Intel's Knight's Landing processor used as compute nodes in DOE supercomputers at LANL, NERSC, and ANL, or for IBM's Power9 processor to be used in the DOE's accelerated supercomputers to be installed at LLNL and ORNL. Processor vendors for future exascale platforms will need to provide appropriate hardware counters and top-down methodologies to support measurement and analysis of CPU performance.

- *Monitoring GPU performance.* How best to understand performance of computations offloaded onto GPU accelerators is an open problem. New mechanisms for monitoring GPU performance have been emerging, but their strengths and weaknesses are not yet entirely understood. After urging by the PIPER team at Rice [2], NVIDIA added support for PC sampling in 2015 to their Maxwell (GM200) GPU [7]. Deficiencies in the initial support for PC sampling were that it reported the total latency of instructions rather than exposed latency and that it only supported very fast sampling rates—at the slowest, one sample per 4096 cycles. After feedback from the PIPER team at Rice, NVIDIA addressed these deficiencies in later versions of their GPUs. NVIDIA's Pascal and emerging Volta GPUs support a wealth of new features, including unified address spaces with CPUs. While we don't yet have enough experience with monitoring features on these chips to understand their weaknesses, we do know that the inability to collect calling contexts on GPUs make it harder to attribute performance to complex GPU code. Ongoing engagements with NVIDIA are focused on obtaining additional software support for attributing performance measurements, e.g., DWARF information that relates machine instructions to inlined call chains.

- *Monitoring data movement.* Monitoring and attributing the cost of data movement is difficult in today's systems. We don't yet have enough experience to understand the strengths and weaknesses of new capabilities for monitoring CPU-GPU and GPU-GPU data movement. On CPU platforms, data movement back and forth to memory occurs in logic outside the processor cores known as the "uncore" on Intel processors or the "nest" on IBM processors. The uncore/nest choreographs transfers to memory on behalf of all cores. For that reason, it can be difficult to attribute data movement occuring in the uncore/nest back to machine instructions and program variables. Hardware technologies that would support better attribution of data movement back to application code would help tools provide insights to application developers.

- *Monitoring network performance.* Today's network measurement hardware and tools (e.g., [12, 9]) are designed for use by system's administrators to monitor the health and traffic in a network, not to attribute problems back to application code that causes them. Future systems will need enhancements that enable fine-grain monitoring of communication so that the nature of communication bottlenecks (e.g., congestion at a particular router) can be traced back to root causes, e.g., communication patterns or a bad logical to physical mapping of processes to nodes in an HPC platform. We believe that capabilities for "message-based sampling", analogous to instruction-based sampling in out-of-order CPUs (e.g., [3, 4]) would help provide the needed insights. Exploring strategies for measurement and analysis based on sampling of communication traffic is the subject of future ongoing research at Rice.

- *Monitoring power and energy.* Energy management interfaces such as Intel's Running Average Power Limiting (RAPL) [5] and AMD's Application Power Management (APM) [1] support estimation and management of power consumption and are not designed for fine-grain measurement and attribution of energy or power consumption. A preliminary study at Rice determined that they are not well suited for use by tools to attribute power or energy consumption to code regions. For exascale systems, where power will be a precious commodity, pinpointing inefficient code regions will be important. New hardware mechanisms will be needed to make that possible.

4.2 Software Technology

Performance tools require proper OS support to make hardware performance monitoring capabilities usable. In addition, for proper measurement and attribution of performance, tools need interfaces at other levels of the software stack as well..

- *Operating Systems.* Today, the Linux `perf_events` interface provides access to hardware performance counters. The `perf_events` interface can be used to measure both application and kernel activity. However, if exascale systems won't use a Linux variant, then exascale tools won't benefit from efforts by the Linux community. On Linux, security concerns limit node-wide monitoring, which is useful on HPC platforms where allocation granularity is at the node level. Using `perf_events` we found that support for first-party monitoring of thread blocking is deficient. As part of a collaboration with IBM through the DOE CORAL project, we have been working to get kernel enhancements added to address this deficiency (`https://www.mail-archive.com/linux-kernel@vger.kernel.org/msg1424933.html`).

- *File Systems.* Recording measurement data on exascale platforms will be difficult. File systems for HPC platforms aren't prepared to record one or more files for every thread in massively-parallel applications. In the absence of such support, substantial software effort will be needed by tools to multiplex performance data into a modest number of large files as an application executes and then to demultiplex the data for analysis.

- *Runtime Libraries.* Tools support in runtime systems for multithreaded programming models is needed to help tools bridge the gap between the source-level specification of a program and its implementation. The development of the OMPT performance tools interface aims to address this problem for OpenMP. Further work will be needed to extend this to other programming models that become of interest for exascale platforms.
- *Communication libraries.* Communication libraries could benefit from additional sufficient support for performance observability. While a tool can wrap the interface of a communication library such as MPI and observe the costs associated with messaging operations, only by having developers of these libraries annotate their implementations with states (e.g., marshalling data for a message, attempting to inject a message, unpacking a message, waiting for a message) can a tool begin to help application developers understand the nature of communication costs. As communication libraries (e.g., MPI, GasNet, Global Arrays) evolve for exascale platforms, tool developers will need to engage communication library developers to ensure that appropriate mechanisms are incorporated into these libraries to help tools monitor communication activity. We believe that communication should expose named, implementation-specific software counters that a tool could inspect to understand the librarys performance. For instance, the GASNet library for one-sided communication might maintain a software counter showing RDMA operation initiation attempts and RDMA operations initiated. An MPI library might similarly expose a counter for MPI operations in progress.

For exascale systems, we expect that performance tools will also need to leverage advances in parallel file systems to record measurement data as applications execute as well as data analytics and scalable visualization for analysis of performance measurements. A full investigation of how to employ such technologies effectively was beyond the scope of this project.

5 Products of the Research

5.1 Contributions to Community Standards

5.1.1 OMPT Tools API for OpenMP

The last standalone draft of the OMPT specification is available at `https://github.com/OpenMPToolsInterface/OMPT-Technical-Report`. In October 2016, shortly after the end of this project OMPT was added to the OpenMP standard and released as part of the OpenMP 5.0 Preview 1 in November 2016, which is available at `http://www.openmp.org/wp-content/uploads/openmp-tr4.pdf`

5.1.2 OMPD Debugging API for OpenMP

The working draft of the specification for the OMPD debugger interface is available at `https://github.com/OpenMPToolsInterface/OMPD-Technical-Report`. This specification is being refined based on experiences at LLNL and Rogue Wave as they develop prototype implementations of OMPD support in the OpenMP runtime library and the OPMD shared library that is a debugger plugin. Once the interface is suitably mature, it will be proposed for inclusion in OpenMP 5.0.

5.2 Open Source Software

5.2.1 HPCToolkit

Performance tools technologies developed as part of HPCToolkit at Rice University during the course of the PIPER project are available from the HPCToolkit website (`http://hpctoolkit.org`) and its Github repository (`https://github.com/hpctoolkit`).

5.2.2 OMPT Tools API in LLVM OpenMP

An initial draft of the OMPT performance tools interface, as described in OpenMP TR 2,[1] was committed to the trunk implementation of the LLVM OpenMP runtime system, which is available at `http://openmp.llvm.org`.

Refinements to the OMPT implementation not yet accepted into the upstream LLVM OpenMP repository are maintained at `https://github.com/OpenMPToolsInterface/LLVM-openmp`.

5.2.3 OMPT Test Suite for OpenMP

A prototype set of regression tests for the OMPT interface was developed and made available at `https://github.com/OpenMPToolsInterface/ompt-test-suite`. Collaborators at RWTH Aachen University improved our initial draft tests and released them as part of the OpenMP test suite for LLVM, which available at as part of LLVM's OpenMP implementation at `http://openmp.llvm.org`.

5.3 Technical Communications

5.3.1 Presentations

- Milind Chabbi, Karthik Murthy, Mike Fagan, and John Mellor-Crummey. HPCToolkit: A Tool for Performance Analysis on Heterogeneous Supercomputers. GPU Technology Conference, March 2013.

- Milind Chabbi, Karthik Murthy, Mike Fagan, and John Mellor-Crummey. Critically Missing Pieces on Accelerators: A Performance Tools Perspective. SC '13: Birds of a Feather Session: Critically Missing Pieces in Heterogeneous Accelerator Computing, Pavan Balaji (Organizer). Supercomputing, 2013

- John Mellor-Crummey. Performance Analysis of MPI+OpenMP Programs on Scalable Parallel Systems. SIAM Conference on Parallel Processing. Portland, OR, February 19, 2014.

- John Mellor-Crummey. OpenMP Tools API (OMPT) and HPCToolkit. SC13 OpenMP Birds-of-a-feather session, Denver, CO, November 19, 2013.

- John Mellor-Crummey. HPCToolkit: Sampling-based Performance Tools for Leadership Computing, Argonne Training Program on Extreme-Scale Computing (ATPESC), August 2014.

- John Mellor-Crummey. Improving Performance Attribution for Optimized Code. Petascale Tools Workshop, Madison Wisconsin, August, 2014.

- John Mellor-Crummey. Introduction to Correctness and Performance Tools for Parallel Programming, SC14 Workshop: Experiencing HPC For Undergraduates: Introduction to HPC Research, November, 2014.

- John Mellor-Crummey. Application Performance Analysis on Large-scale Parallel Computer Systems BP, December 2014.

- John Mellor-Crummey. Performance analysis of MPI+OpenMP Programs with HPCToolkit. Tutorial at the Rice Oil and Gas HPC Workshop, March 2015.

- John Mellor-Crummey. Performance analysis of MPI+OpenMP Programs with HPCToolkit. HPC Summer Institute, Rice University, June 2015.

[1] Available as `http://www.openmp.org/wp-content/uploads/ompt-tr2.pdf`.

- Laksono Adhianto, HSA Interface: What tool developers want. Heterogeneous System Architecture (HSA) meeting, June 2015.

- John Mellor-Crummey. Missing pieces in the OpenMP ecosystem. International Workshop on OpenMP. Keynote Address. Aachen, Germany, October 2015.

- John Mellor-Crummey. Performance analysis of MPI+OpenMP Programs with HPCToolkit. Tutorial at the Rice Oil and Gas HPC Workshop, March 2016.

- John Mellor-Crummey and Mark Krentel. Low-overhead Monitoring of Parallel Applications. Mini-Symposium 10: Improving Performance, Throughput, and Efficiency of HPC Centers through Full System Data Analytics. SIAM Conference on Parallel Processing for Scientific Computing. Paris, France. April 2016.

5.3.2 Publications

Journal Articles

- Ashwin M. Aji, Lokendra S. Panwar, Feng Ji, Karthik Murthy, Milind Chabbi, Pavan Balaji, Keith R. Bisset, James Dinan, Wu-chun Feng, John Mellor-Crummey, Xiaosong Ma, Rajeev Thakur, "MPI-ACC: Accelerator-Aware MPI for Scientific Applications", IEEE Transactions on Parallel & Distributed Systems, vol. 27, no. , pp. 1401-1414, May 2016, DOI: `https://doi.org/10.1109/TPDS.2015.2446479`

Conference Papers

- Xu Liu and John Mellor-Crummey. A data-centric profiler for parallel programs. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. (SC '13). November 2013, Denver, CO, USA. IEEE. DOI: `http://dx.doi.org/10.1145/2503210.2503297`

- Alexandre Eichenberger, John Mellor-Crummey, Martin Schulz, Michael Wong, Nawal Copty, Robert Dietrich, Xu Liu, Eugene Loh, Daniel Lorenz. OMPT: An OpenMP Tools Application Programming Interface for Performance Analysis. In: Rendell A.P., Chapman B.M., Mller M.S. (eds). OpenMP in the Era of Low Power Devices and Accelerators. IWOMP 2013. Lecture Notes in Computer Science, vol 8122. Springer, Berlin, Heidelberg. DOI: `https://doi.org/10.1007/978-3-642-40698-0_13`.

- Milind Chabbi, Karthik Murthy, Michael Fagan, John Mellor-Crummey. Effective sampling-driven performance tools for GPU-accelerated supercomputers. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. (SC '13). November 2013, Denver, CO, USA. IEEE. DOI: `http://dx.doi.org/10.1145/2503210.2503299`.

- X. Liu and J. Mellor-Crummey. Pinpointing data locality bottlenecks with low overhead. In 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 183–193, April 2013, Austin, TX, USA. IEEE. DOI: `http://dx.doi.org/10.1109/ISPASS.2013.6557169`.

- Xu Liu, John Mellor-Crummey, and Michael Fagan. 2013. A new approach for performance analysis of openMP programs. In Proceedings of the 27th international ACM conference on International conference on supercomputing (ICS '13). ACM, New York, NY, USA, 69-80. DOI: `https://doi.org/10.1145/2464996.2465433`.

- Ashwin Aji, Lokendra Panwar, Feng Ji, Milind Chabbi, Karthik Murthy, Pavan Balaji, Keith R. Bisset, James Dinan, Wu-chun Feng, John Mellor-Crummey, Xiaosong Ma, and Rajeev Thakur. 2013. On the efficacy of GPU-integrated MPI for scientific applications. In Proceedings of the 22nd international Symposium on High-performance Parallel and Distributed Computing (HPDC '13). ACM, New York, NY, USA, 191-202. DOI: `http://dx.doi.org/10.1145/2462902.2462915`.
- Xu Liu, Kamal Sharma, and John Mellor-Crummey. ArrayTool: a lightweight profiler to guide array regrouping, Proceedings of the 23rd International Conference on Parallel Architectures and Compilation, August 24-27, 2014, Edmonton, AB, Canada. DOI: `https://doi.org/10.1145/2628071.2628102`.
- Milind Chabbi, Xu Liu, and John Mellor-Crummey. Call Paths for Pin Tools. In Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO '14). ACM, New York, NY, USA, Pages 76 , 11 pages. DOI: `http://dx.doi.org/10.1145/2544137.2544164`.
- Xu Liu and John Mellor-Crummey. A tool to analyze the performance of multithreaded programs on NUMA architectures. In Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '14). ACM, New York, NY, USA, 259-272. DOI: `http://dx.doi.org/10.1145/2555243.2555271`.
- Milind Chabbi, Michael Fagan, and John Mellor-Crummey. 2015. High performance locks for multi-level NUMA systems. In Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2015). ACM, New York, NY, USA, 215-226. DOI: `https://doi.org/10.1145/2688500.2688503`.
- Milind Chabbi, Wim Lavrijsen, Wibe de Jong, Koushik Sen, John Mellor-Crummey, and Costin Iancu. 2015. Barrier elision for production parallel programs. In Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2015). ACM, New York, NY, USA, 109-119. DOI: `https://doi.org/10.1145/2688500.2688502`.
- Milind Chabbi and John Mellor-Crummey. 2016. Contention-conscious, locality-preserving locks. In Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '16). ACM, New York, NY, USA, Article 22, 14 pages. DOI: `https://doi.org/10.1145/2851141.2851166`.
- Sri Raj Paul, M Araya-Polo, J Mellor-Crummey, D Hohl. Performance Analysis and Optimization of a Hybrid Seismic Imaging Application. Procedia Computer Science 80, 8-18, 2016. DOI: `https://doi.org/10.1016/j.procs.2016.05.293`.
- Laksono Adhianto, Philip Taffet. Addressing Challenges in Visualizing Huge Call-Path Traces. Proceedings of the 6th International Workshop on Parallel Software Tools and Tool Infrastructures. (PSTI '16). ICPP Workshops. August 2016, Philadelphia, PA, USA. IEEE. Pages 319–328. DOI: `https://doi.org/10.1109/ICPPW.2016.53`.

Theses

- Xu Liu. Performance Analysis of Program Executions on Modern Parallel Architectures. Ph.D. Thesis, Department of Computer Science, Rice University, July 2014. `http://hdl.handle.net/1911/87790`.

- Milind Chabbi. Software Support for Efficient Use of Modern Computer Architectures. Ph.D. Thesis, Department of Computer Science, Rice University, August 2015. `http://hdl.handle.net/1911/87730`.

- Sri Raj Paul. Performance Analysis and Optimization of a Hybrid Seismic Imaging Application. M.S. Thesis, Department of Computer Science, Rice University, February 2016.

5.3.3 Reports

- Alexandre Eichenberger, John Mellor-Crummey, Martin Schulz, Nawal Copty, Jim Cownie, Robert Dietrich, Xu Liu, Eugene Loh, Daniel Lorenz, and other members of the OpenMP Tools Working Group. OMPT: An OpenMP Tools Application Programming Interface for Performance Analysis. OpenMP Technical Report 2. April 2014. Available as `http://www.openmp.org/wp-content/uploads/ompt-tr2.pdf`.

- Alexandre Eichenberger, John Mellor-Crummey, Martin Schulz, Nawal Copty, John DelSignore, Robert Dietrich, Xu Liu, Eugene Loh, Daniel Lorenz, and other members of the OpenMP Tools Working Group. OMPT and OMPD: OpenMP Tools Application Programming Interfaces for Performance Analysis and Debugging OpenMP Technical Report. April 24, 2013. Available as `http://www.openmp.org/wp-content/uploads/ompt-tr.pdf`

5.3.4 Videos

- John Mellor-Crummey. Measuring and Attributing Performance of Applications that Employ Emerging Template-based Parallel Porgramming models. Video. X-stack Meeting, April 2016. `http://www.cs.rice.edu/~johnmc/x-stack/piper-video-hpctoolkit-xstack-2016.mp4`

- John Mellor-Crummey. Understanding the Performance Characteristics of PGAS Codes. Video. X-stack Meeting, April 2016. `http://www.cs.rice.edu/~johnmc/x-stack/degas-hpctoolkit-xstack-2016.mp4`

References

[1] Advanced Micro Devices. BIOS and Kernel Developers Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors, Jan. 2013. Available as `http://developer.amd.com/wordpress/media/2012/10/42301_15h_Mod_00h-0Fh_BKDG1.pdf`.

[2] M. Chabbi, K. Murthy, M. Fagan, and J. Mellor-Crummey. Critically missing pieces on accelerators: A performance tools perspective. SC '13: Birds of a Feather Session: Critically Missing Pieces in Heterogeneous Accelerator Computing, Pavan Balaji (Organizer). Supercomputing, 2013. `http://www.hpctoolkit.org/pubs/SC_2013_BOF_MilindChabbi.pdf`. November 20, 2013.

[3] J. Dean, J. E. Hicks, C. A. Waldspurger, W. E. Weihl, and G. Chrysos. ProfileMe: hardware support for instruction-level profiling on out-of-order processors. In *MICRO 30: Proceedings of the 30th annual ACM/IEEE International Symposium on Microarchitecture*, pages 292–302, Washington, DC, USA, 1997. IEEE Computer Society.

[4] IBM Corporation. Power ISA Version 2.07B. April 9, 2015.

[5] Intel. Intel 64 and IA-32 Architectures Software Developers Manual, 2016. Order Number: 253669-060US. Available as `https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf`.

[6] D. Levinthal. Gooda PMU event analysis package. `https://github.com/David-Levinthal/gooda`.

[7] S. Matwankar. CUDA 7.5: Pinpoint performance problems with instruction-level profiling. `https://devblogs.nvidia.com/parallelforall/cuda-7-5-pinpoint-performance-problems-instruction-level-profiling`. September 8, 2015.

[8] M. Srinivas, B. Sinharoy, R. J. Eickemeyer, R. Raghavan, S. Kunkel, T. Chen, W. Maron, D. Flemming, A. Blanchard, P. Seshadri, J. W. Kellington, A. Mericas, A. E. Petruski, V. R. Indukuru, and S. Reyes. IBM POWER7 performance modeling, verification, and evaluation. *IBM Journal of Research and Development*, 55(3):4:1–4:19, May 2011.

[9] H. Subramoni, A. M. Augustine, M. Arnold, J. Perkins, X. Lu, K. Hamidouche, and D. K. Panda. *INAM^2: InfiniBand Network Analysis and Monitoring with MPI*, pages 300–320. Springer International Publishing, Cham, 2016.

[10] N. Tallent and J. Mellor-Crummey. Effective performance measurement and analysis of multithreaded applications. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*, Raleigh, North Carolina, USA, February 2009.

[11] N. Tallent, J. Mellor-Crummey, and A. Porterfield. Analyzing lock contention in multithreaded applications. In *PPoPP '10: Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 269–280, New York, NY, USA, 2010. ACM.

[12] M. Technologies. Unified Fabric Manager software product brief. `http://www.mellanox.com/related-docs/prod_management_software/PB_UFM_Software.pdf`.

[13] A. Yasim. A top-down method for performance analysis and counters architecture. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 35–44, March 2014.