Final Report

Center for Scalable Application Development Software

Project period: January 2007–December 2012

Cooperative Agreement No. DE-FC02-07ER25800

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

The Center for Scalable Application Development Software (CScADS) was established as a partnership between Rice University, Argonne National Laboratory, University of California Berkeley, University of Tennessee – Knoxville, and University of Wisconsin – Madison. CScADS pursued an integrated set of activities with the aim of increasing the productivity of DOE computational scientists by catalyzing the development of systems software, libraries, compilers, and tools for leadership computing platforms. Principal Center activities were workshops to engage the research community in the challenges of leadership computing, research and development of open-source software, and work with computational scientists to help them develop codes for leadership computing platforms.

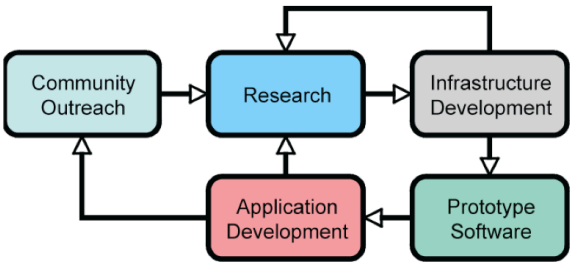

Figure 1: Relationship between CScADS activities.

Figure 1 illustrates the relationships between the Center's activities. The flow of ideas originated from two sources: (1) workshops for community outreach and vision-building, and (2) direct engagement with application teams. These activities helped us identify important problems that merited further research. The desire to tackle research challenges we identified drove the development of software insfrastructure to support the research. The resulting infrastructure supported not only the research, but also construction of prototype software to support application development. Finally, experiences applying emerging system software, libraries, compilers, and tools to leadership computing challenges spurred the next cycle of research and development. In the following sections, we briefly summarize the CScADS activities at Rice University in these areas.

2 Community Outreach and Vision Building

To engage the community in the challenges and foster interdisciplinary collaborations, the CScADS center organized an annual series of 3-4 day workshops focused on topics related to scalable software for the DOEs leadership computing platforms. Goals for the workshops included:

- identification of important open problems and challenges for achieving high performance on leadership computing systems,
- brainstorming on promising approaches to open problems,
- identification of infrastructure needs to address key challenges and assessment of available infrastructure,
- identification of opportunities for synergy, opportunities to consolidate and harden existing infrastructures, opportunities to reuse components developed by others, as well as opportunities to refactor and extend existing components to apply them to new challenges,
- collaboration on design of sharable components,
- interaction and information interchange between computer scientists and SciDAC application teams, and

- identification of targets of opportunity for further investment of resources, in particular strategic investment targets for the DOE Office of Science.

To meet these goals, CScADS organized workshops of two kinds:

- *Enabling technology workshops.* These workshops brought together computer science researchers to exchange ideas about technologies important to the DOE mission of harnessing the power of petascale platforms for scientific computing, and collaborate on the development of these technologies. Attendees at these workshops included representatives from leading academic research groups, and key players in the commercial space.

- *Outreach Workshops.* At these workshops, members of the CScADS center and their partners provided training for representatives from SciDAC and INCITE application teams about hardware and software for scientific computing at the petascale, best practices for leadership computing, as well as scientific data analysis & visualization. These workshops included hands-on sessions in which computer scientists and application scientists collaboratively explored application challenges on leadership computing platforms and the role of enabling technologies to address them.

Attendance at the workshops was by invitation only. Attendees included a mix of senior researchers, post-docs, and graduate students. The workshops typically had between 20–40 attendees. Below, provide a short description of the themes for each of the series of workshops, list the years in which the workshop was held, and describe some outcomes of each workshop series.

2.1 Enabling Technology Workshops

2.1.1 Workshop on Libraries and Algorithms for Petascale Applications

This workshop, held in 2007, brought together computer scientists working on algorithms and libraries with members of the SciDAC application teams. The principal workshop goal was to identify challenges for library and algorithm developers from the needs of the SciDAC applications and to foster collaboration between the communities. Workshop topics included the use of multicore processors and the use of automatic tuning in libraries. SciDAC application developers who attended the workshop learned about the state of the art in terms of what numerical libraries are available, how best to use them, and how to make them work well on the various systems. In turn, library developers who attended the workshop were able to get a better understanding of what SciDAC applications need from libraries in terms of functionality, interface design, and algorithms.

One outcome of this workshop was a substantial improvement in I/O scaling and performance of the Omega3P simulation tool being developed at Stanford Linear Accelerator Center. Discussions in the hands-on session at the workshop led to use of collective communication patterns to avoid scaling bottlenecks associated with reading input data. Additionally, adjusting the application to use parallel netCDF and MPI-IO reduced the time for writing output data by a factor of 100 when Omega3P was run on thousands of processors on the Cray XT system at Oak Ridge. These improvements dramatically enhanced the scalability of Omega3P.

2.1.2 Workshop on Automatic Tuning for Petascale Systems

This workshop series, held 2007–2008, brought together researchers from different fields who share a common interest in systems that automatically tailor scientific codes to their target machines. The workshop included researchers and practitioners in automatic tuning, compiler code generation, and architecture design. The aim of the workshop was to identify opportunities and challenges for using automatic tuning on future petascale systems.

The workshop focused on exploring strategies for optimizing code to achieve high performance on the complex node architectures found on today's leadership computing platforms. In recent

years, nodes have grown in complexity, including multicore processors that have larger core counts with each generation, accelerators, and deep memory hierarchies to mask the ever-growing gap between on-chip computation performance and off-chip memory access performance. Each of these features requires a significant effort in tuning of applications and libraries. As a result, compiler and library developers have turned to automating the process of software tuning, using large amounts of computation time to explore a space of different variants of the program and running each variant on the target architecture.

Talks presented in the workshop series covered a wide range of subjects, including programming models and practices for current and anticipated architectures, numerical libraries, novel algorithms, program generators, and compiler-based techniques that support either auto-tuning, auto-parallelization, or both. Interspersed with the technical talks were periods set aside for discussions. Attendees broke up into small groups to discuss narrow topics of interest, such as how to achieve high performance with important inner loops, or specific strategies for tuning MPI parameters. Plenary discussions explored topics of broad interest, such as how to build shared infrastructure and how to bring together the multiple autotuning communities.

2.1.3 Workshop on Performance Tools for Extreme-Scale Computing

This workshop series, held 2007–2012, brought together a collection of researchers interested in building performance and correctness tools for leadership computing platforms.

Leadership computing platforms installed at Argonne National Laboratory and Oak Ridge National Laboratory represent a dramatic increase in scale and complexity compared to previous parallel systems. Making effective use of systems at this scale requires performance tools that help application developers measure, analyze, attribute, and understand application performance bottlenecks. Challenges include recording, analyzing, and presenting performance data from many nodes, as well as developing new techniques that afford insight into the performance of complex node architectures. At the node level, challenges include understanding the performance of multithreading on multicore processors and understanding the efficiency of hybrid parallelism on compute nodes with one or more attached accelerators. Providing tools that make it possible to address these problems requires increasingly sophisticated methods for instrumentation, measurement, analysis and modeling of application performance. Meeting these challenges at scale requires tools with unprecedented capabilities.

The goal of this workshop series was to bring together tools researchers to discuss challenges of performance analysis and debugging on emerging petascale systems, review ongoing research, and work as a community to tackle the challenges of performance analysis on petascale systems. Attendees at the workshop were tools developers who work on tool infrastructure, debugging, performance instrumentation, measurement, analysis, and visualization. About a third of each workshop was allocated for attendees to work together in small groups to tackle specific problems of shared interest.

Specific aims of the workshop series included:

- identifying performance tool capabilities needed to analyze the spectrum of issues that arise with emerging extreme scale systems,
- discussing emerging capabilities of research and commercial tools in this space,
- identifying common needs, functionality, and opportunities for sharing infrastructure,
- discussing design aspects of sharable components, and
- developing standardized interfaces to facilitate sharing.

A central focus of this workshop series has been to promote development of sharable software components to accelerate the development of performance tools for leadership computing platforms. The motivation for this effort comes from the fact that the software infrastructure needed to address measurement, analysis, and modeling of application performance on petascale systems is too large for any one group to develop alone. This effort to catalyze collaboration among the tools community led to groups carving out pieces of their tools and repackaging them as sharable components. The resulting components are now used by multiple groups.

One particularly important product of this workshop series was a preliminary design for OMPT—a new performance tools API for OpenMP. OpenMP 4.0 is viewed as best available programming model for node architectures for emerging parallel systems. Today, OpenMP lacks a standard interface for performance tools, which poses a significant obstacle to building high-quality, multiplatform tools that provide insight into the performance of applications that exploit OpenMP at the node level. OMPT has since been recognized by the OpenMP Architecture Review Board as an official technical report. We will continue to work to see that OMPT makes it into the OpenMP language standard.

2.1.4 Workshop on Libraries and Autotuning for Extreme-Scale Applications

This workshop series, held 2009–2012, brought together researchers and practitioners in automatic tuning, in library design and construction, and in compiler-based code generation to identify and discuss opportunities and challenges for tailoring code to current and future extreme-scale platforms using automatic tuning. Attendees at the workshop included members of the compiler and library autotuning communities, as well as attendees who work on runtime optimization of scientific programs.

Over the last decade, microprocessor features such as deep pipelines, multiple cores, and complex memory hierarchies have made it increasingly difficult to achieve good performance in scientific applications and libraries. This fact has given rise to systems that automate the tuning process, using large amounts of computation to configure the application or library for good performance on the target architecture.

The workshop consisted of a series of talks and discussion sessions. Most workshop attendees presented talks on their research, their insights, and their experiences. Discussion sessions focused on opportunities to build shared infrastructure, along with issues raised during the workshops.

2.2 Outreach Workshops

2.2.1 Leadership Computers, Petascale Applications, and Performance Strategies

This workshop series, held from 2007–2012, had the explicit purpose of facilitating the use of large-scale computational resources by DOE-funded projects. To that end, announcements were sent to the principal investigators of all SciDAC projects and INCITE awards, asking them to nominate as attendees individuals in their projects who were, or would be working directly with the project's code and running it on the leadership class platforms. This approach worked well and all available slots at the workshop were filled with attendees who had the responsibility for making their codes scale up and run well on the largest machines. The goals of the workshop were that the attendees:

- become familiar with the architecture, operation, and usability issues for each of the DOE leadership computing facilities, understand application scaling bottlenecks on the systems,
- learn strategies for achieving good performance with message passing and I/O libraries,
- explore new programming models, languages, and techniques that can provide scalable performance, and

- learn the tools and strategies for understanding the performance of petascale applications.

We began each of these workshops by having each of the application team representatives give a short presentation about their project with an emphasis on their code and the scaling challenges it faces. In the afternoon, representatives of the three Office of Science computing facilities (ANL, ORNL, LBNL) gave presentations about their leadership computing platforms. DOE center staff were present at the workshop to facilitate getting accounts, logging in, and coping with any system administration issues that arose for workshop attendees using the leadership computing platforms.

These workshops continued with tutorials about MPI, parallel I/O, OpenMP, and performance tools. An important aspect of these tutorials is describing proven strategies for high-performance programming. Workshops also included talks about CAF and UPC. Unscheduled time provided an opportunity for participants to work directly with workshop presenters and organizers to obtain hands-on experience applying what they have learned to their own applications. Many of the participants chose to continue collaborative work in "late-night hacking" sessions after dinner. Feedback from the participants about the workshops was overwhelmingly positive. Participants were particularly enthusiastic about hands-on time working directly with library and tools experts.

After attending one of the workshops, several participants went on to submit successful proposals for INCITE awards.

2.2.2 Scientific Data Analysis and Visualization for Extreme-Scale Computing

This workshop series, held 2008–2012, explored computer science topics at the intersection of storage, analysis, and data-intensive computational science. Traditionally, computation, storage, and analysis were viewed as three separate tasks, but as supercomputers and data increase in size and complexity, this isolated view is no longer practical. By considering these topics together, the aim of this workshop was to uncover potential efficiency and scalability, and see the connections between these areas.

Within this broad scope, topics discussed at the workshops included data models, run-time and post-processing analysis and visualization techniques, I/O systems and optimizations, data mining and machine learning, and the application of all of the above in science codes. Talks at the workshop discussed solutions that are ready to use today, such as Paraview, VisIt, and VisTrails., as well as research expected to yield results 1-5 years out. In addition to talks by the participants, the workshop series included unstructured time for attendees to discuss possible collaborations and to apply visualization tools to their datasets. This approach was appreciated by the attendees, who used the time to get tools installed and try them out, build readers that could import their datasets into the production tools, and discuss expected challenges in upcoming simulation runs.

3 Summary Description of the Research Performed

Research in CScADS explored a range of software technologies to improve the productivity of application developers building high-performance codes for leadership computing platforms. These technologies included languages, compilers, runtime systems, and tools. The principal foci of our research were (1) performance tools for scalable parallel systems, and (2) language and compiler technology for high performance computing. In the following sections, we briefly summarize our principal work in these areas.

3.1 Performance Tools for Scalable Parallel Systems

With principal support from CScADS and supplemental support from the Performance Engineering Research Institute, as part of the DOE SciDAC-2 program, we developed HPCTOOLKIT, an integrated suite of tools for sampling-based performance analysis of parallel codes on systems ranging from laptops to leadership computing platforms.

HPCTOOLKIT provides accurate measurement of a program's use of resources and waste, attributes the usage to source lines in a program, works with multilingual, fully optimized binaries, has very low overhead and scales to thousands of processors. We have deployed HPCTOOLKIT on systems ranging from clusters to supercomputers at DOE leadership computing facilities. In the sections below, we describe the key capabilities for application performance analysis developed as part of the HPCToolkit project under SciDAC-2 funding, and and highlight the accomplishments of our research and development.

3.1.1 HPCTOOLKIT Key Capabilities

HPCTOOLKIT works by sampling each thread in a parallel program using an interval timer or hardware performance counter as a trigger, unwinding the call stack and attributing the performance metric associated with the sample event to the program's calling context where the sample occurs. Below we list some of the key strengths of HPCTOOLKIT's approach.

Scalable measurement and analysis. For performance tools to be useful on extreme-scale parallel systems, measurement and analysis techniques must be scalable. HPCTOOLKIT's asynchronous sampling of thread activity is scalable, does not require inter-process communication, and the sampling rate can be tuned to keep overhead arbitrarily low. Profile data for threads is compact. For computations involving enormous numbers of cores, HPCTOOLKIT supports sampling a subset of the cores to help keep executions from producing an excess of data.

Binary analysis for attributing costs to optimized code. HPCTOOLKIT combines both static and dynamic (runtime) analysis of program binaries to measure the performance of fully optimized, multilingual programs, including binary-only libraries, and attribute these measurements to source code (where available) [Tallent et al. PLDI09]. A strength of HPCTOOLKIT is its ability to unwind the call stack of an application executing optimized code. To do this, HPCTOOLKIT uses on-the-fly binary analysis to determine how to unwind the call stack so that we can attribute costs to the full calling context in which they are incurred. In addition, HPCTOOLKIT introduces new techniques for analyzing program binaries to recover information about loops, and inlined code correlate them with the original source.

Blame shifting for root cause analysis. HPCTOOLKIT introduces a new kind of technique for identifying the root causes of performance loss and inefficiency in parallel codes; we call this approach *blame shifting*. To date, we have employed this strategy to two problems: understanding lock contention [Tallent et al. PPoPP10] and pinpointing performance losses caused by insufficient parallelism and parallelization overhead [Tallent and Mellor-Crummey PPoPP09; Tallent and Mellor-Crummey Computer09] for the multithreaded Cilk parallel language. The blame shifting approaches are supported by *problem-focused* measurement that is specifically designed to quantify and attribute a particular kind of losses. To provide a feeling for how this class of techniques works, consider the problem of lock contention. When a thread is spin waiting for a lock, time it spends waiting is a *symptom* of lock contention. Knowing that there is spin waiting without knowing who is causing it is of only limited help. We shift the blame for idleness due to lock contention from the waiting thread to the lock holder by (a) charging the cost of samples incurred by the waiting thread to the lock, and (b) having a thread assume the blame for idleness associated with a lock as it thread releases it. We accomplish this blame shifting for unmodified optimized codes by injecting a wrapped version of locking primitives that perform the accounting we need. For multithreading, we shift the blame of spin waiting for work to threads that have work but aren't sharing it [Tallent and Mellor-Crummey PPoPP09; Tallent and Mellor-Crummey Computer09].

Managing complexity with top-down analysis. To make analysis of large programs and parallel executions tractable, HPCTOOLKIT presents performance analysis in a top-down fashion, that

helps users immediately identify what is important and help them drill down to quickly locate the cause of abottlenecks. HPCTOOLKIT's call path profiling enables it to attribute costs to the full calling contexts in which they are incurred. HPCTOOLKIT's user interfaces support hierarchical presentations of performance data in both profiles and traces.

Associating performance measurements with source code. HPCTOOLKIT's `hpcviewer` [Adhianto et al PSTI10] user interface relates performance metrics to program source code. `hpcviewer` offers three main views. The Calling Context view is a top-down view showing dynamic calling contexts and their costs. The Callers view is a bottom-up view for looking upward along call paths. This view is especially useful for understanding how costs in a function are incurred from multiple contexts. The Flat view presents costs associated with a program's static structure with costs summed over all contexts. All views present both inclusive and exclusive metrics. Exclusive metrics only reflect costs for a scope itself; inclusive metrics reflect costs for the entire subtree rooted at that scope.

Figure 2 shows an example of `hpcviewer`'s display of the PFLOTRAN code run on 8184 cores on a Cray XT5. In this example, the Calling Context view in the bottom pane shows the hot path from the `pflotran` main function to PETSc's `PCApplyBAorAB` along with metrics for load imbalance and total cycles. The top pane (two lines in this example) shows the source call to `PCApplyBAorAB`. In between are two scatter plots of the time spent (by MPI rank) in `PCApplyBAorAB` and a sibling function in the call tree, `VecDotNorm2`. The figure explains load imbalance here in the program. Low times in `PCApplyBAorAB` by some processes are offset by their high times in `VecDotNorm2`. Investigation of child contexts reveals that the processes that finish early in `PCApplyBAorAB` end up waiting in `VecDotNorm2`.

Pinpointing and quantifying scaling losses. As the number of cores increases, inefficiencies due to scaling losses become more of a problem. Programs that run efficiently at a few thousand cores may not run well at a million cores. HPCTOOLKIT pinpoints scaling losses within and across nodes in parallel systems. Furthermore, it quantifies losses and attributes them to program contexts using only node-level profile data. For example, in a weak scaling performance study, one might run a program on 4K cores and again on 32K cores. If the program scaled perfectly, we would expect the node level costs to be constant with weak scaling. By differencing the call path profiles of two executions (see [Coarfa et al ICS07; Tallent et al. SC09]) any value above zero represents a loss beyond perfect scaling. HPCTOOLKIT associates scaling losses with the full calling context in which they occur. Losses can be identified with a few clicks in HPCTOOLKIT's `hpcviewer` interface to precisely pinpoint the cause of the scaling inefficiency.

Supporting analysis using derived metrics. Although raw metrics can identify program hot spots, effective tuning requires an understanding of how and where resources are used *inefficiently*. Derived metrics such as the difference between peak and actual performance are far more useful than raw metrics. For instance, computing the difference between peak FLOPs possible (total cycles * maximum FLOPs per cycle) and measured FLOPs in every program context quantifies and pinpoints missed opportunities for higher performance. Such derived metrics can be computed by entering a spreadsheet-like formula in HPCTOOLKIT's `hpcviewer` user interface and the results can be explored interactively.

Providing scalable views of activity over time. HPCTOOLKIT's `hpctraceviewer` [Tallent et al. ICS11] visualization tool shows how a parallel execution unfolds over time. During execution, HPCTOOLKIT collects a sequence of call stacks via asynchronous sampling for each process or thread and `hpctraceviewer` displays these call stacks across the lifetime of the process. `hpctraceviewer` allows rapid top-down performance analysis by displaying traces at multiple levels

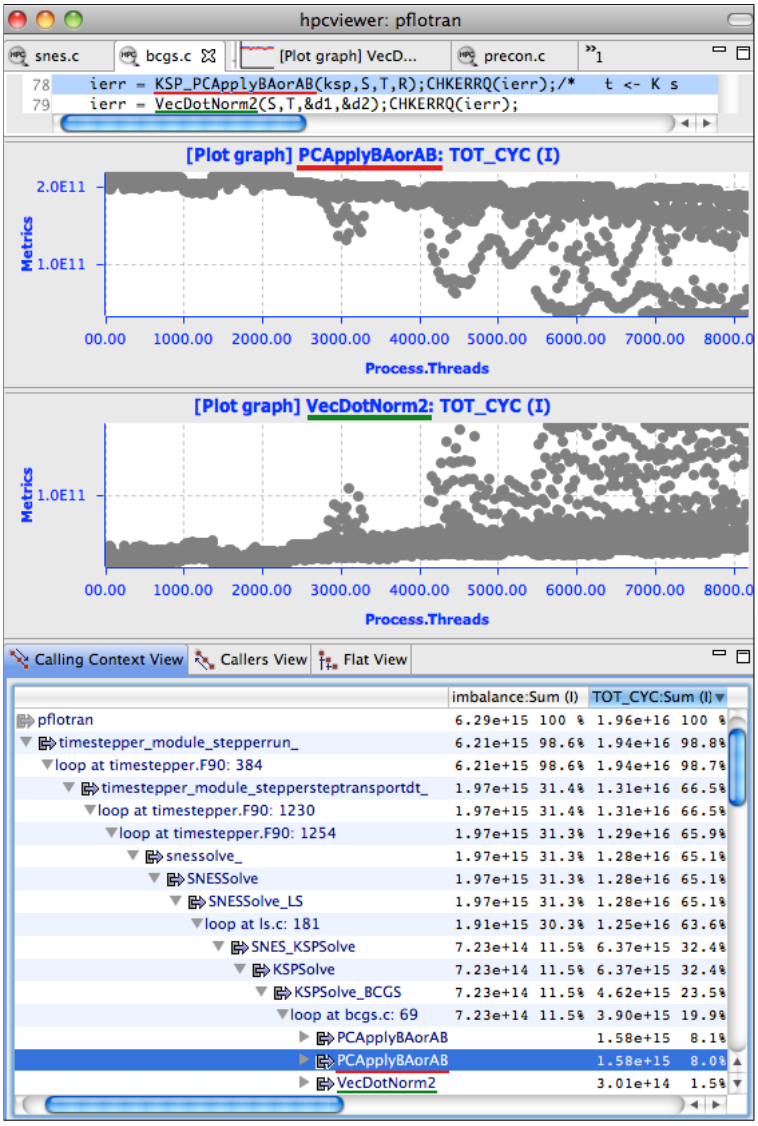

Figure 2: A Calling Context view of PFLOTRAN's load imbalance.

of abstraction. Regardless of execution scale, the cost for rendering an `hpctraceviewer` visualization is proportional to the product of the number of pixels on the display and the log of the number of trace records per thread.

Figure 3 shows an example `hpctraceviewer` display of a PFLOTRAN execution taking 982 seconds on 8184 cores on a Cray XT5. The three main panes are the Trace view (top left), the Call Path view (top right) and the Depth view (bottom left). The Trace view shows a user-controllable slice in the process/time/call-path space, where process rank in on the vertical axis and time moves left to right on the horizontal axis. In this example, we see that at depth 3, PFLOTRAN alternates between two phases (purple and black). Depth 6 shows that these two phases use the solver (tan) and depths 7 and 14 show that they use the solver in different ways.

The Call Path view shows the full call path at the current point in the Trace view, and the Depth view shows one process's call depth over time. The user can zoom in on the Trace view in both the process and time dimensions and can see the overall process execution at any call depth.

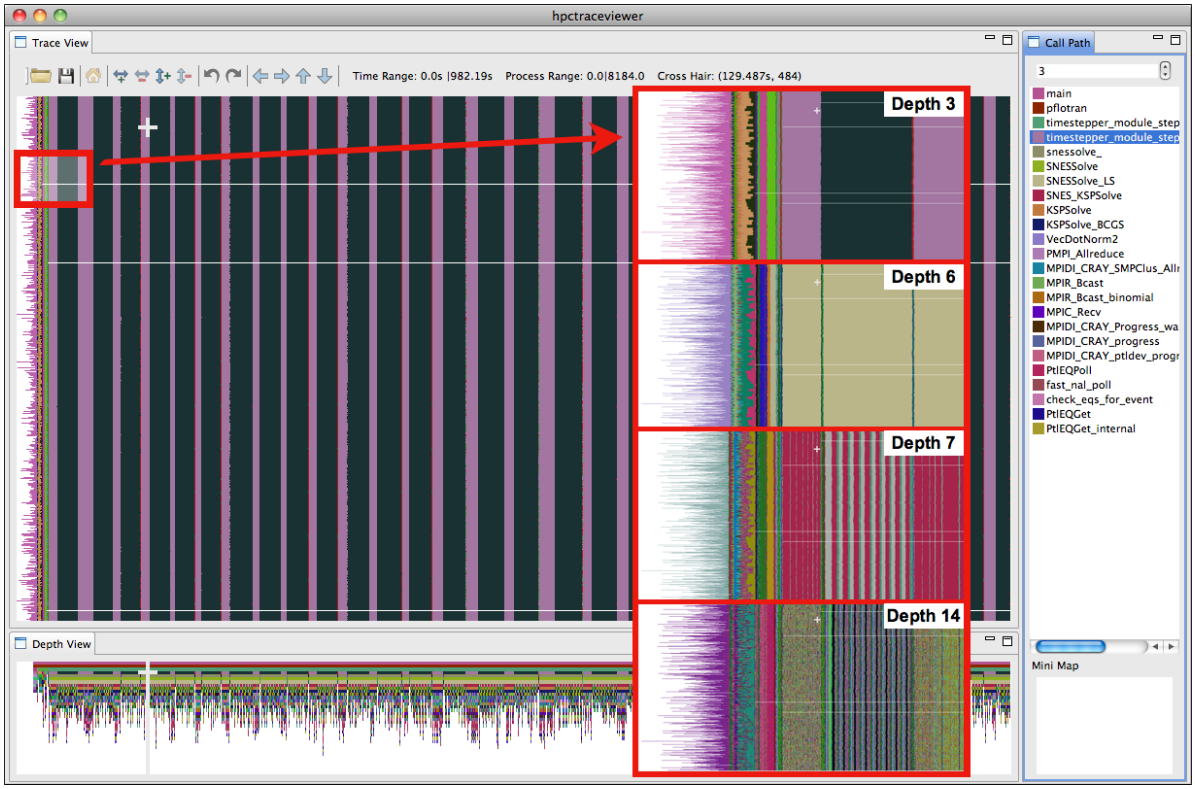

Figure 3: An 8184-core execution of PFLOTRAN on a Cray XT5. The inset exposes call path hierarchy by showing the selected region (top left) at different call path depths.

3.1.2 Key Accomplishments of Performance Tools Research

With support from both CScADS and the Performance Engineering Research Institute to build tools to analyze application performance on parallel systems, we accomplished a lot over the course of the SciDAC2 program. Here, we list the key accomplishments in the HPCToolkit project.

- We engaged IBM and Cray to pinpoint and address kernel problems that prevented performance monitoring using asynchronous sampling on Blue Gene/P and Cray XT/XE6 leadership computing platforms. As a result of our efforts, sampling-based performance analysis is now possible on DOE leadership computing platforms.

- We engaged the broader community (Linux kernel developers, hardware performance monitor device driver authors, UTK developers of the PAPI library) to address problems with sample-based performance monitoring using hardware counters.

- We developed `libmonitor`—a process control layer that supports measurement of multiple multithreaded processes in a programming model independent fashion. `libmonitor` works with MPI, OpenMP, Pthreads, Coarray Fortran, Unified Parallel C, Global Arrays and compositions thereof.

- We developed a wide spectrum of novel techniques to support measurement, analysis, and attribution of costs measured using asynchronous sampling, including

  – on-the-fly binary analysis to attribute costs measured using asynchronous sampling to full calling contexts in optimized applications [Tallent et al. PLDI09]; and

  - an interactive tool for presenting calling-context-sensitive performance metrics in ways that rapidly focus an analyst's attention on performance bottlenecks [Adhianto et al. PSTI10]; and
  - a differential profiling technique for quantifying scalability losses within and across nodes on leadership computing systems [Coarfa et al. ICS07; Tallent et al. SC09]; and
  - new problem-focused techniques for measurement and attribution of performance problems in parallel applications, including techniques for
    * quantifying performance losses in multithreaded computations due to insufficient parallelism & parallel overhead and attributing them to source code responsible [Tallent and Mellor-Crummey PPoPP09; Tallent and Mellor-Crummey Computer10]; and
    * quantifying performance losses due to idling that results from lock contention in multithreaded programs; attributing them to calling contexts holding the locks [Tallent et al. PPoPP10];
  - a strategy for quantifying and pinpointing performance losses in scalable parallel codes resulting from load imbalance [Tallent et al. SC10]; and
  - an approach measuring memory latency with hardware performance counters and attributing it to both code regions and data objects [Liu and Mellor-Crummey CGO11]; and
  - a technique for producing insight into transient behavior of parallel programs at multiple levels of abstraction by collecting and analyzing traces of asynchronous call path samples [Tallent et al. ICS11].

- We have been active in transferring our technology. Both Bull and SciCortex deployed HPCToolkit on their systems for their customers.

3.2 Languages and Compiler Technology for High Performance Computing

Within CScADS, research on compiler technology for high performance computing focused on compiler and runtime systems for partitioned global address languages, compiler optimization to improve node performance of scientific codes, in addition to analysis and optimization of scripting languages. We briefly summarize research and development activities in each of these three areas.

3.2.1 Partitioned Global Address Space Languages

The Partitioned Global Address Space (PGAS) model, exemplified by the UPC, Coarray Fortran, and Titanium Languages enables programmers to easily express parallelism on complex shared data structures. Work in CScADS and the Center for Programming Models for Scalable Parallel Computing worked to make implementations of PGAS languages efficient and available.

Coarray Fortran Language and Implementation. In a partnership with the Center for Programming Models for Scalable Parallel Computing, we worked to develop a second-generation compiler for Coarray Fortran. To enable CScADS to deliver a more robust source-to-source Coarray Fortran compiler that could be used by application scientists, we worked with LLNL's Rose compiler team to develop software infrastructure to support a source-to-source Coarray Fortran compiler based on Rose. The focus of CScADS efforts in this area was to work with LLNL to complete Rose's language support for Fortran. As part of this work, we added support for Fortran modules to support separate compilation of Fortran programs. This work included synthesis of module interface specifications, adding support to import interfaces from external modules, and creating regression tests for Rose's Fortran support. As part of this work, we added support for

Coarray to LANL's OpenFortran parser, which is used as the Fortran front-end by Rose, along with support for source-to-code generation in Fortran 90.

The Coarray Fortran 2.0 (CAF 2.0) programming model developed as a product of our research is a partitioned global address space programming model based on one-sided communication. The design for CAF 2.0 goes well beyond the 1998 design of Coarray Fortran (CAF) by Numrich and Reid. CAF 2.0 is a coherent synthesis of concepts from MPI, Unified Parallel C, and IBM's X10 programming language. CAF 2.0 includes a broad array of features including process subsets known as teams, team-based asynchronous collective communication, communication topologies, dynamic allocation of shared data, and global pointers, along with synchronization constructs including finish, a communication fence, and events.

We used CAF 2.0 to implement the High Performance Computing Challenge (HPCC) benchmarks, including High Performance Linpack (HPL), RandomAccess, Fast Fourier Transform (FFT), and STREAM triad. On 4096 CPU cores of a Cray XT with 2.3 GHz single socket quad-core Opteron processors, we achieved 18.3 TFLOP/s with HPL, 2.01 GUP/s with RandomAccess, 125 GFLOP/s with FFT, and a bandwidth of 8.73 TByte/s with STREAM triad [Jin et al. IPDPS11]. In 2010, CAF 2.0 was recognized with the *Class II Award: Most Productive Language.* at the HPC Challenge Awards Competition held at SC10. In 2011, it was recognized with *Class II Honoable Mention: Performance and Productivity* at the HPC Challenge Awards Competition held at SC11.

Because of our experience with Coarray Fortran, we were approached to critique a proposal by the Fortran J3 Standards Committee to incorporate support for coarrays and synchronization into Fortran 2008. We carefully reviewed the details of the proposal and provided detailed white paper (Fortran J3 paper 08-126) to the Standards Committee. Our white paper led the standards committee to remove several ill-considered features from the Fortran 2008. specification. The influence of CAF 2.0 continues today: the Fortran standards committee is considering new features based on CAF 2.0 primitives.

Synchronization for PGAS Languages As part of an effort to make PGAS languages more expressive and simplify programming, Rice explored using software transactional memory (STM) as a mechanism for supporting synchronization using atomic operations. For software transactional memory to be practical for use with a programming model such as Coarray Fortran, it must be very efficient. One drawback of using STM is the cost of validating transactional reads.

Timestamp-based validation techniques for STM significantly reduce the cost of validations by reducing the number of unnecessary validations. These techniques can be optimistic (allowing a transaction to proceed even though a conflict is possible, with the hope that the transaction will still be able to commit), or pessimistic (aborting the transaction as soon as a possible conflict is detected). For any given static workload and contention scenario the choice between a pessimistic and optimistic strategy can result in significant performance differences, while in an application where the workload and contention vary throughout the execution neither of these techniques will result in optimal performance.

We developed a runtime tuning strategy that uses on-the-fly monitoring to determine the most effective validation technique for a given state of the STM system. Our hybrid validation strategy adaptively chooses between optimistic and pessimistic validation depending on the state of the STM system. We evaluated our technique on a set of standard STM benchmarks and demonstrated that our strategy performs within a couple of percent of the best validation strategy for a given static workload scenario, and that it outperforms both optimistic and the pessimistic validation techniques by up to 18% in long-running, dynamically-changing scenarios [Zhang et al. SPAA 2008; Zhang et al. EPHAM 2008].

Extensions to C for Parallel Programming. We extended the ROSE optimizing compiler infrastructure from LLNL to add lightweight task parallelism constructs to an extension to the C language developed at Rice University called Habanero-C. In addition to adding new language constructs to ROSE's handling of the C language, we have also modified the build process for the ROSE infrastructure to support pre-compilation of all non-essential (from a standpoint of a compiler writer) libraries that come with ROSE, allowing for much faster incremental builds in an optimizing compiler infrastructure such as Habanero-C.

3.2.2 Compiler Technology for Optimizing Node Performance

Source-to-source optimization for improving memory hierarchy performance. Motivated by node performance problems with S3D—a Fortran code that performs direct numerical simulation of turbulent combustion being developed at Sandia National Laboratory, CScADS researchers at Rice University extended LoopTool—a compiler-based tool that helps expert human programmers improve the performance of Fortran loop nests by applying complex patterns of transformations to tailor the loop nests for a target microprocessor. LoopTool automates the application of well-known source-to-source transformations that improve data reuse at various levels of the memory hierarchy, adjust instruction mix, and generate code that can be scheduled more efficiently by a conventional Fortran compiler.

To use LoopTool, one takes a Fortran procedure and annotates the code with directives that specify a transformation recipe. LoopTool then applies the recipe to perform both the explictly-specified transformations along with other supporting transformations that need not be specified explicitly. To support optimization of key loops in Sandia National Laboratorys S3D code, we extended LoopTool with support for the transformations described below.

- *Scalarization of Fortran 90 Array Syntax.* Scalarization transforms a computation specified using Fortran 90 array notation into a loop nest that iterates over each element in the index space and performs the computation elementwise.

- *Loop unswitching.* Unswitching a loop means hoisting a conditional within a loop nest out of one or more levels of enclosing loops and creating a custom version of the loop nest for the true and false branches of the conditional. By creating condition-free loop bodies, unswitching enables instructions to be scheduled more effectively. Unswitching was added to LoopTool specifically to support optimization of S3Ds diffusive flux computation. This transformation enables a loop nest to be written in a natural fashion a single copy of code with embedded loop-invariant conditionals and have LoopTool generate custom copies of the loop nest tuned for each setting of flag variables that may arise during execution.

In addition, LoopTool's support for multi-level loop fusion and unroll-and-jam transformations was enhanced to support application to loops generated by scalarizing Fortran 90 array syntax. In FY08, Rice issued a subcontract to Texas State to create a hardened Linux version of Looptool that could be distributed as open source software that can be used by application teams to help address performance problems identified in their codes. That effort produced an open source version of LoopTool for Linux. LoopTool was used to improve the performance of memory intensive loop nests in S3D and LoopTool-generated code was incorporated into the production version of S3D.

Dynamic optimization of complex applications. Sophisticated scientific applications are often assembled out of a diverse set of components constructed by different software teams. The Common Component Architecture (CCA) was devised to aid in assembling such applications. A problem with component-based approaches to software is that abstraction boundaries between components can be costly. Today, avoiding excessive costs at component boundaries forces developers to write

coarser-grain components than they might like. CScADS has been exploring the use of dynamic program optimization for CCA applications, namely analyzing and optimizing software at run time when all component interfaces are bound and full information about the bindings is available. To explore the potential for optimizations at this level, we experimented with interprocedural optimization of SIDL and Babel code, which serves as the interface between components in CCA applications, using the LLVM compiler infrastructure. Tests using the TTSTT mesh benchmarks from the Center for Component Technologies in Terascale Simulation Software showed the promise of this approach. With interprocedural optimization, we were able to reduce the overhead of fine-grain operations from 5.5x to 3.5x. Much of the remaining overhead is due to the cost of allocating temporary objects. Experiments with a fast allocator dropped the overhead to 2.5x. Further work would have be needed to extend this preliminary work and turn out a tool capable of performing such optimizations on production applications.

New techniques for redundancy elimination. CScADS researchers at Rice developed a novel technique for detecting redundant computations where the redundancy occurs on different iterations of a loop under different names. These redundancies arise routinely in array address calculations and computations on array-element values (such as stencil or wave front techniques). To increase the transformations effectiveness, he also developed a new approach to algebraic re-associationusing commutativity and associativity to rearrange expressions in ways that expose additional opportunities for optimization [Cooper et al. PACT08].

3.2.3 Compiler Analysis and Optimization of Scripting Languages

The difficulty of developing sophisticated high performance parallel applications is well known. This DOE Office of Science, the NNSA, and the NSA have been struggling with aspects of this problem for years. Better software technologies to accelerate development of high performance applications for leadership computing platforms would be welcomed by the application teams. At the FY07 CScADS Summer Workshop on Libraries and Algorithms for Petascale Applications, application teams indicated that they would prefer to program in high-level scripting languages. The popularity of scripting languages among developers of scientific application has been growing year after year. To support this goal, CScADS began to explore compiler and run-time technology to make it practical to use scripting languages for high performance computing. Rice began to explore a range of compiler technologies to accelerate each of the major scripting languages used for scientific computing, namely, Matlab, R, and Python.

Matlab. Work on Matlab compilation focused on exploring interprocedural type inference to enable application developers to write general library routines and have these routines be automatically specialized for the contexts in which they are used. Work primarily concentrated on hardening existing implementations of type analysis and specialization to support experimental evaluation using non-trivial codes. After the untimely death of Ken Kennedy and the departure of his graduate students working in this area, work on this topic ceased.

R. Unlike Matlab, in R bindings for procedures and variables are lexically scoped and one cannot determine which procedure will be called at a call site without flow-sensitive interprocedural analysis. Also, R procedure calls use call-by-need binding, which means that expressions passed as actual arguments are passed as unevaluated expressions. As a result, R code is typically interpreted. Significant sources of run-time overhead are function and variable name resolution.

To compile R into efficient code, Rice worked to generalize the OpenAnalysis compiler infrastructure to support analysis of R programs. This work included generalizing the call graph representation in OpenAnalysis to facilitate analysis of programs with late binding of function variables, to support control flow graph construction for lexically-scoped programs, and to support

for data-flow analysis of lexically-scoped programs. This infrastructure was developed to support compilation of scripting languages such as R into native code, have functions resolved at compile time rather than run-time, and transform Rs interpreted call-by-need procedure arguments into call-by-value arguments evaluated by compiled code.

Performance analysis of generated code for R showed that lack of data reuse in the memory hierarchy seriously impaired runtime performance. Garbage collected languages generally have poor locality as they churn through the heap allocating fresh data. To address this problem, researchers at Rice explored interprocedural analysis techniques to understanding variable lifetimes so that heap-based storage allocation could be converted into more efficient stack-based allocation. The aim of this approach was to improve memory hierarchy performance by increasing locality. Based on our analyses, we explored analysis to enable a our compiler to stack-allocate data, which would improve locality and lead to data reuse in cache. The approach pursued was to compile R programs into calls to library functions that were the equivalent of actions by the R interpreter and then attempt to reason about the resulting code to optimize its memory performance. This approach was inspired by Ken Kennedy's vision to build compilers to optimize library-based programming models. Ultimately, we found that analysis of the library-based code was too difficult because of information loss and this effort was abandoned.

Python. For Python, we explored a different approach. Our thesis was that we could build an effective compiler for Python by translating it into a statically typed functional language that has a good compiler and automatic memory management. In particular, modern statically-typed languages provide precise control over data representations, and come with runtime systems that have competitive performance. To investigate the viability of this approach, we built a compiler for Python by translating it into OCaml. An interesting practical advantage of using modern statically-typed functional languages is that they use Hindley-Milner type systems, which means that there is no need for the translation to construct type terms. We compared the performance of our implementation, Monty, with that of CPython, the reference Python implementation, and with Jython, a Java implementation of Python, using a suite of 370 benchmarks. Our experiments show that some programs compiled using our approach run up to 4.6 times faster than CPython. However, for engineering reasons, some programs also run significantly slower than CPython. We pinpointed causes of performance degradation and assessed the potential for removing these causes in future work. Our implementation is significantly faster than Jython, up to a factor of 100 in some cases. A by product of this research was a proposal for an improved array copying implementation in OCaml. These results are reported in a Ph.D. dissertation [Bandyopadhyay 2009].

After investing the efforts of several graduate students and obtaining few results that looked like they would have near-term practical impact, we made the decision to refocus our efforts on other topic areas.

4 Application Engagement

As part of CScADSs engagement efforts, Rice University worked closely with several of the SciDAC application teams to diagnose application performance bottlenecks on leadership-class platforms using a combination of measurement, analysis, and modeling. In particular, Rice worked closely with the SciDAC S3D and GTC application teams. These collaborations began with participation in engagement activities associated with Tiger Teams of the Performance Engineering Research Institute. This involvement has been useful in (1) identifying interesting computer science problems that are important for application teams and (2) using preliminary analysis of applications to guide tool development so that tools directly address the problems of critical importance. Below, we

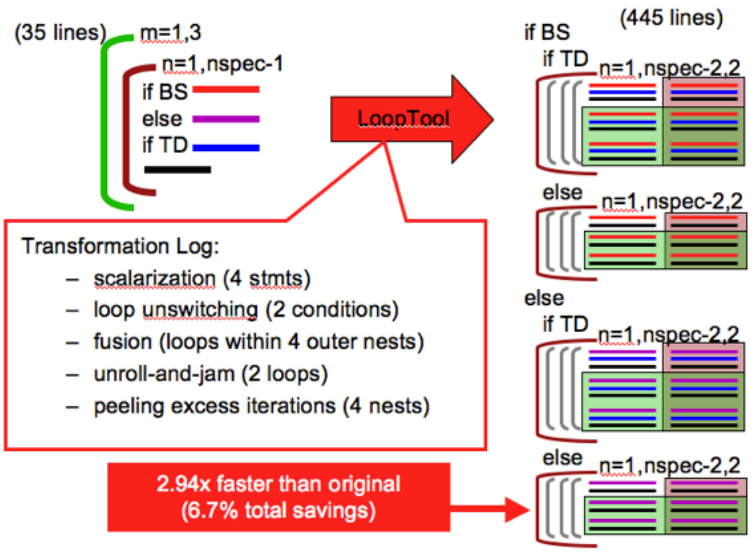

Figure 4: Applying LoopTool to S3Ds diffusive flux computation.

briefly describe some of our work with S3D and GTC application summarize interactions with other SciDAC and INCITE enabling technology and application efforts.

4.1 S3D

Analysis of S3D using Rice University's HPCToolkit performance tools uncovered opportunities for using source-to-source tools to tailor code to improve memory hierarchy utilization. This led to refinement of Rice's LoopTool program transformation tool. Applying LoopTool to S3D yielded improved performance of S3D's diffusive flux calculation(most memory intensive loop) by nearly a factor of three [Mellor-Crummey SciDAC07]. Figure 4 shows a cartoon that depicts the transformation of S3D's diffusive flux calculation.

Additionally, analysis of experiments with S3D on the hybrid Cray XT3/XT4 system showed that the lower memory bandwidth on the XT3 nodes hurt the weak scaling performance of S3D on the hybrid system. Further analysis showed that performance on the hybrid system could be improved by proportionally adjusting the partitioning of computation to account for the higher efficiency of the XT4 nodes. A paper describing S3D along with the aforementioned work to tailor it to ORNL's Jaguar appeared in Computational Science & Discovery in 2009 [Chen et al. CS&D09].

4.2 GTC

One of the most significant application engagement activities undertaken by CScADS reserachers at Rice was analysis and tuning of the Gyrokinetic Toroidal Code (GTC)—a SciDAC-funded code being developed to study the impact of fine-scale plasma turbulence on energy and particle confinement in the core of tokamak fusion reactors. The GTC code is a centerpiece of the International Thermonuclear Experimental Reactor project and as part of the DOEs INCITE program, it was awarded millions of processor hours on the Cray XT4 at ORNL.

Work with GTC included exploration of opportunities for improving memory hierarchy utilization. One component of this effort has been studying the impact of data structure layout and code organization on the spatial and temporal locality present in data access patterns. In FY07, a detailed study of GTC using a performance modeling toolkit developed at Rice identified several opportunities for improving application performance. These included reorganizing the particle data

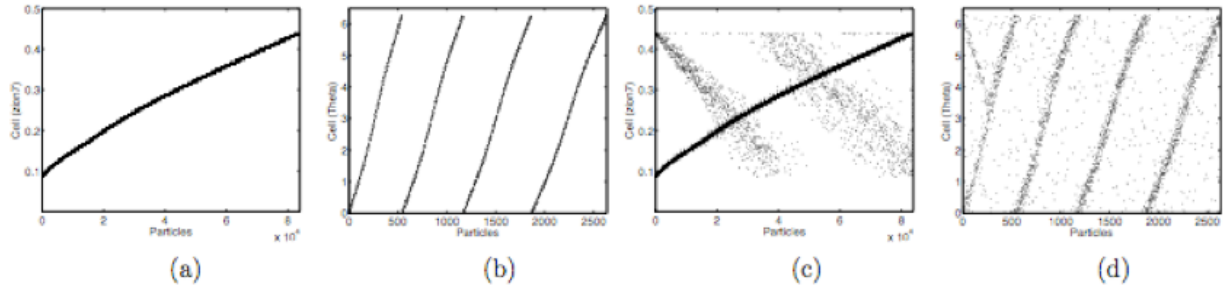

Figure 5: Particle positions become disordered as execution progresses in GTC.

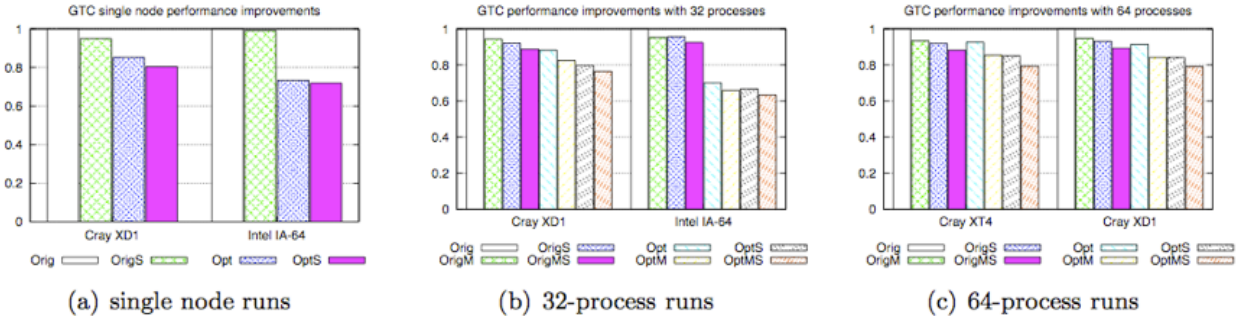

Figure 6: Performance of GTC with different code optimizations.

structures to improve spatial reuse in the charge deposition and particle pushing phases of the application, using loop fusion to increase temporal reuse of particle data, and transforming the code to increase instruction-level parallelism and reduce translation look-aside buffer misses. Overall, these code transformations improved performance by 33%. Code modifications were provided back to the application team. In FY08, CScADS researchers at Rice determined that as a GTC simulation progresses, the particles become increasingly disordered with respect to the underlying grid cells representing the volume within the tokamak. Charge deposition and particle position updates involve interactions between particles and cells. As particles become disordered with respect to the cells, these phases get slower due to increasingly inefficient use of the memory hierarchy in each compute node.

In Figure 5, we show the particle positions along the radial and θ directions at the time steps 0 and 20. To address this problem, we augmented GTC to periodically sort particles by their cell number during the simulation. Each sorting step restores locality and performance. However, sorting is not free. Maximizing performance requires determining the appropriate number of time steps between sorting operations. We were able to formulate this as a minimization problem and adaptively compute the proper interval between sorting operations. A paper describing this work was presented at SciDAC 2008 [Marin et al. SciDAC 2008].

We compared up to eight versions of the GTC codes on the three parallel systems (including a Cray XT4 at LBL), with three types of optimizations turned on and off. In naming the versions, *Orig* represents the original version of the code, *Opt* includes loop and data restructuring optimizations by Rice in FY07, *M* includes an improved implementation of particle migration between processes that preserves locality that was developed at Rice in FY08, and *S* represents the adaptive sorting approach developed by Rice in FY08.

Figure 6(a) presents the single node performance results of three optimized versions of the GTC codes on a Cray XD1 and an Intel Itanium 2. All timing results are normalized to the execution time of the original version. Overall, the version with all optimizations applied achieves the best performance with a 27% reduction of execution time on the Itanium2 cluster.

Figures 6(b) and (c) compare the parallel execution time of all eight versions of the code. In Figure 6(b), the optimized version *OptMS* with all three optimizations applied achieved the best performance with about 37% reduction of overall execution time on the Itanium2 cluster. The loop and data restructuring optimization contributed significantly to the overall improvement on the Itanium 2 cluster. In Figure 6(c), the best version of the code with all three optimizations applied achieves approximately a 21% execution time reduction from the original version on two Opteron-based Cray machines.

4.3 Other Application Engagement Activities

Helping application and enabling technology teams with performance analysis of their software has been the principal mode of interaction between CScADS and application teams. Below we mention a few significant interactions that inspired new tools research in CScADS and publications about that work.

- **MFDn.** The "Many Fermion Dynamics nuclear" code being developed by the UNEDF SciDAC project evaluates the many-body Hamiltonian and obtains the low-lying eigenvalues and eigenvectors using the Lanczos algorithm. While the code shows good scaling and load balance on 15,000 cores, it is lacking in per-process efficiency. A study of the code on Opteron processors using HPCToolkit indicated that roughly 20% of the lost opportunity for efficiency arose from use of a compressed-sparse column format within a sparse-matrix vector multiply [Tallent et al. SciDAC 2008].

- **Chroma.** Chroma is a C++ code from the US Lattice QCD project. The codes extensive use of C++ expression templates motivated development of new capabilities for binary analysis in the HPCToolkit performance tools. The difficulty of understanding performance measurements for Chromas expression templates motivated integration of static and dynamic information in the presentation of call path profiles in HPCToolkits hpcviewer interface. The aforementioned work on HPCToolkit received the distinguished paper award at the ACM Symposium on Programming Language Design and Implementation [Tallent et al. PLDI09].

- **Madness.** In FY09, the HPCToolkit team at Rice partnered with PERI-supported researchers Robert Fowler and Allan Porterfield at UNC and began to work directly with Robert Harrison on MADNESS, which was selected as an early science code for ORNL's Cray XT5. Motivated by understanding the performance issues in MADNESS, the HPCToolkit team developed new measurement and analysis techniques to pinpoint performance bottlenecks in multithreaded codes and implemented them in the HPCToolkit performance tools. This research not only resulted in feedback to the MADNESS team about issues causing performance losses in their multithreaded code, but also development of a new technique for attributing resource contention [Tallent et al. PPoPP10].

- **Chombo.** Analysis of Chombo with a tool developed to detect dead writes identified opportunities for optimization based on redundant initialization [Chabbi and Mellor-Crummey 2012].

5 Products of the Research

5.1 Sampling-based Performance Monitoring on DOE Computing Platforms

Delivering sampling-based performance tools for all DOE computing platforms, especially the leadership computing platforms, had been a principal goal of Rice's HPCToolkit project work as part of CScADS. Our extensive testing of operating system support for interrupt-driven profiling on the full range of DOE platforms (including Cray, Blue Gene, and Linux clusters) uncovered critical bugs on all systems that rendered profiling unusable. Rice researchers engaged the community (including developers of hardware performance counter device drivers, the community of Linux kernel developers, kernel developers at Cray and IBM, and the PAPI team at Tennessee) to fix these problems and provided regression tests. This work had several positive outcomes.

- *Linux clusters.* Support for sampling-based performance monitoring is now a standard part of the Linux kernel.

- *Cray supercomputers.* Support for sampling-based performance monitoring has been become a standard feature on Cray supercomputers since Compute Node Linux 2.1 in December 2008.

- *Blue Gene supercomputers.* Support for sampling-based performance monitoring became available on Blue Gene/P platforms with the V1R3M0 compute kernel release installed at ANL in January 2009. CScADS researchers at Rice contributed to the Argonne SOW, including requirements for hardware and software support for sampling-based performance and an extended set of regression tests used to evaluate the performance monitoring support as part of the Blue Gene/Q acceptance test. Deep engagement with IBM as part of the procurement process helped shape the design of the Blue Gene Performance Monitoring (BGPM) interface and led to the IBM delivering appropriate support for sampling-based performance monitoring on Blue Gene/Q platforms.

5.2 Open Source Software for Performance Tools

- Released an open source implementation of the HPCToolkit performance tools. HPCToolkit includes libmonitor - a tool harness for application control, an infrastructure for measurement and analysis of application performance data, hpcviewer - a user interface that supports interactive exploration of profiles of parallel applications, and hpctraceviewer - a user interface that supports interactive analysis of execution traces of parallel applications collected using sampling. HPCToolkit has been installed on leadership-class systems at ORNL, ANL, and NERSC, NSF supercomputers at TACC, as well as supercomputers in Australia, China, Norway, Switzerland, and the UK, to name just a few.

5.3 Open Source Software for Compilers

- Contributed open source code for Fortran semantic analysis to the ROSE compiler infrastructure. Our work included adding support for modules and separate compliation. ROSE won an R&D 100 award in 2009.

- Released an open source implementation of Coarray Fortran 2.0 based on the ROSE compiler infrastructure.

- Released an open source implementation of LoopTool - a source to source transformation tool that performs sophisticated dependence-based transformations for memory hierarchy optimization, including loop alignment, fusion, unroll-and-jam, tiling, and iteration space splitting.

5.4 Impact on Standards

- Fortran 2008. Research on Coarray Fortran 2.0 influenced the Fortran 2008 standard.[1] Extensions currently being considered for Fortran 2008 were inspired by primitives developed as part of the Coarray Fortran 2.0 project.

- OMPT performance tools API for OpenMP. One tangible product of research on performance tools for multithreaded node programs is an emerging performance tool API for OpenMP known as OMPT. The design for OMPT began at the 2012 *CScADS Workshop on Performance Tools for Extreme-scale Computing*. Today, the OMPT interface has been approved by the OpenMP Architecture Review Board as an OpenMP technical report. The aim is to incorporate the design for OMPT as part of the OpenMP language standard.

6 Technical Communications

6.1 Theses

- Adam Bordelon, Developing A Scalable, Extensible Parallel Performance Analysis Toolkit. M.S. Thesis, Department of Computer Science, Rice University, May 2007.

- Apan Qasem, Automatic Tuning of Scientific Applications. Ph.D. Thesis. Department of Computer Science, Rice University, July 2007.

- Nathan R. Tallent. Performance Analysis of Optimized Code: Binary Analysis for Performance Insight. MS Thesis, Department of Computer Science, Rice University, 2008.

- Rajarshi Bandyopadhyay, Compiling dynamic languages via statically typed functional languages. Ph.D. Thesis, Rice University, 2009.

- Nathan R. Tallent, Performance Analysis for Parallel Programs: From Multicore to Petascale. Ph.D. Thesis, Department of Computer Science, Rice University, 2010.

6.2 Papers

- Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John MellorCrummey, Nathan R Tallent. HPCToolkit: Tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience, Special issue on Software Technology for High-End Computing. 22(6):685-701, April 2010.

- Laksono Adhianto, John Mellor-Crummey, and NathanR. Tallent. Effectively presenting call path profiles of application performance. In PSTI 2010: Workshop on Parallel Software Tools and Tool Infrastructures, in conjunction with the 2010 International Conference on Parallel Processing, pages 179–188, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

- Milind Chabbi, John Mellor-Crummey, and Keith Cooper. Efficiently exploring compiler optimization sequences with pairwise pruning. In ACM SIGPLAN 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era, San Jose, CA, June 2011.

- Milind Chabbi and John Mellor-Crummey. 2012. DeadSpy: a tool to pinpoint program inefficiencies. In Proceedings of the Tenth International Symposium on Code Generation and Optimization (CGO '12). ACM, New York, NY, USA, 124-134.

[1]Work on Coarray Fortran 2.0 was jointly supported by the Center for Programming Models for Scalable Parallel Computing.

- J. H. Chen, A. Choudhary, B. R. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using S3D, Computational Science & Discovery, 2 015001, 2009.
- Cristian Coarfa, John Mellor-Crummey, Nathan Froyd, and Yuri Dotsenko. Scalability analysis of SPMD codes using expectations. In Proceedings of the International Conference on Supercomputing, Seattle, WA, June 2007.
- Keith Cooper, Jason Eckhardt and Ken Kennedy. Redundancy Elimination Revisited, Seventeenth International Conference on Parallel Architectures and Compilation Techniques (PACT 08), Toronto, Canada, October 2008.
- R. Fowler, L. Adhianto, B de Supinski, M. Fagan, T. Gamblin, M. Krentel, J. Mellor-Crummey, M. Schulz and N. Tallent. "Frontiers of performance analysis on leadership-class systems." In Proceedings of the SciDAC 2009 conference, J. Phys.: Conf. Ser. 180 012041 (6pp) DOI: 10.1088/1742-6596/180/1/012041.
- Guohua Jin, Laksono Adhianto, John Mellor-Crummey, William N. Scherer III, and Chaoran Yang. Implementation and performance evaluation of the HPC challenge benchmarks in Coarray Fortran 2.0. In IPDPS 11: International Symposium on Parallel and Distributed Systems, Anchorage, Alaska, May 2011.
- Xu Liu and John Mellor-Crummey. Pinpointing data locality problems using data-centric analysis. In CGO 11: Proceedings of the 2011 International Symposium on Code Generation and Optimization, Chamonix, France, April 2011.
- XuLiu, John Mellor-Crummey, and NathanR. Tallent. Analyzing application performance bottlenecks on Intel's SCC. Proc. of the TACC-Intel Highly Parallel Computing Symposium, 2012.
- Gabriel Marin and John Mellor-Crummey. "Pinpointing and Exploiting Opportunities for Enhancing Data Reuse", In Proceedings of the 2008 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'08), Austin, TX, April 2008.
- Gabriel Marin and John Mellor-Crummey. Application insight through performance modeling. In Proceedings of the 26th IEEE International Performance, Computing, and Communications Conference, New Orleans, LA, USA, April 2007.
- Gabriel Marin, Guohua Jin, and John Mellor-Crummey 2008 Managing Locality in Grand Challenge Applications: A Case Study of the Gyrokinetic Toroidal Code, Proceedings of the SciDAC 2008 conference, J. Phys.: Conf. Ser. 125 012087. IOP Publishing, August 2008.
- John Mellor-Crummey. Harnessing the power of emerging petascale platforms. In Proceedings of the SciDAC Conference 2007. Journal of Physics: Conference Series 78 (2007) 012048.
- J. Mellor-Crummey, L. Adhianto, and W. Scherer III. A New Vision for Coarray Fortran. The 3rd Conference on Partitioned Global Address Space Programming Models, Ashburn, VA. October 5-8 2009.
- John Mellor-Crummey, Peter Beckman, Keith Cooper, Jack Dongarra, William Gropp, Ewing Lusk, Barton Miller, and Katherine Yelick. Creating Software Tools and Libraries for Leadership Computing, CTWatch. October 2007.

- John Mellor-Crummey, Peter Beckman, Jack Dongarra, Barton Miller, and Katherine Yelick. Software Technology for Leadership-class Computing. SciDAC Review 5. Fall 2007, pp. 36-45.

- J. Sandoval and Keith Cooper. Dynamic Compilation for Component-Based High-Performance Computing. The 2009 Workshop on Component-Based High Performance Computing (CBHPC 2009), November 15-16 2009, Portland, Oregon, USA. (Extended Abstract.)

- Nathan R. Tallent and John Mellor-Crummey. Using sampling to understand parallel program performance. In Holger Brunst et al., editor, Tools for High Performance Computing 2011. Springer-Verlag, 2012.

- Nathan Tallent, John Mellor-Crummey, Laksono Adhianto, Michael Fagan, and Mark Krentel 2008. HPCToolkit: performance tools for scientific computing, Proceedings of the SciDAC 2008 conference, J. Phys.: Conf. Ser. 125 012088.

- Nathan Tallent and John Mellor-Crummey. Effective performance measurement and analysis of multithreaded applications. In Proc. of Symposium on the Principles and Practice of Parallel Programming (PPoPP), 229-240, Raleigh, NC, February 2009.

- N. Tallent and J. Mellor-Crummey, Identifying Performance Bottlenecks in Work-Stealing Computations. IEEE Computer. Special issue on Tools and Environments for Multi- and Many-Core Architectures. December 2009.

- Nathan Tallent, John Mellor-Crummey, Laksono Adhianto, Michael Fagan, and Mark Krentel 2008. HPCToolkit: performance tools for scientific computing, Proceedings of the SciDAC 2008 conference, J. Phys.: Conf. Ser. 125 012088.

- N. Tallent, J. Mellor-Crummey, L. Adhianto, M. Fagan, and M. Krentel. Diagnosing Performance Bottlenecks in Emerging Petascale Applications. Proceedings of Supercomputing 2009 (SC09), Portland, OR, USA, November, 2009.

- N. Tallent, J. Mellor-Crummey, and M. Fagan. Binary analysis for measurement and attribution of program performance. In Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation (Dublin, Ireland, June 15 - 21, 2009). PLDI '09. ACM, New York, NY, 441-452. Distinguished Paper Award.

- N. Tallent, J. Mellor-Crummey, and Allan Porterfield. Analyzing Lock Contention in Multithreaded Applications. In Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (Bangalore, India January 11 - 13, 2010). PPoPP '10. ACM, New York, NY.

- Nathan R. Tallent, John M. Mellor-Crummey, Michael Franco, Reed Landrum, and Laksono Adhianto. Scalable fine-grained call path tracing. In Proc. of the 25th International Conference on Supercomputing, pages 6374, New York, NY, USA, 2011. ACM.

- Nathan R. Tallent, Laksono Adhianto, and John M. Mellor-Crummey. Scalable identification of load imbalance in parallel executions using call path profiles. In SC '10: Proc. of the 2010 ACM/IEEE Conference on Supercomputing, pages 111, Washington, DC, USA, 2010. IEEE Computer Society.

- Rui Zhang, Zoran Budimlic and William N. Scherer III. Commit Phase in Timestamp-based STM. SPAA 2008: 20th ACM Symposium on Parallelism in Algorithms and Architectures, June 2008.

- Rui Zhang, Zoran Budimlic, William N. Scherer III and Mackale Joyner. Runtime Tuning of STM Validation Techniques. Workshop on Exploiting Parallelism with Transactional Memory and other Hardware Assisted Methods (EPHAM). April 2008.

6.3 Presentations

- L. Adhianto. A New Vision for Coarray Fortran. The 3rd Conference on Partitioned Global Address Space Programming Models, Ashburn, VA. October 6, 2009.

- M. Fagan, The Long and (Un)Winding Road. Paradyn/Dyninst Week, University of Maryland, College Park, Maryland, April 27-28, 2009.

- M. Fagan, HPCToolkit Update 2009. Center for Scalable Application Development Software's Workshop on Performance Tools for Petascale Computing. Tahoe City, CA, USA, July 20-23, 2009.

- J. Mellor-Crummey, Programming Languages/Models and Compiler Technologies, Microsoft Manycore Workshop, Seattle, WA, June 21, 2007. Overview talk as panel chair.

- J. Mellor-Crummey, Harnessing the power of emerging petascale platforms, DOE Scientific Discovery though Advanced Computing (SciDAC) Annual Meeting, Boston, MA, June 28, 2007. Invited talk.

- J. Mellor-Crummey, Sampling-based Strategies for Measurement and Analysis, Center for Scalable Application Development Systems (CScADS) Workshop on Performance Tools for Petascale Computing. Snowbird, UT. July, 2007.

- J. Mellor-Crummey, Measurement, Analysis, and Modeling of Parallel Programs, Dagstuhl Seminar on Code Instrumentation and Modeling for Parallel Performance Analysis. Dagstuhl, Germany. August, 2007. Invited Talk.

- J. Mellor-Crummey. Sampling-based Application Performance analysis on the Cray XT System using HPCToolkit, Cray Booth, SC07, November 2007.

- J. Mellor-Crummey. HPCToolkit Components, HPCSW Workshop on Open—Speedshop, Denver, CO, March 2008.

- J. Mellor-Crummey. Fine-tuning your HPC investments with Application Performance Analysis, Oil and Gas HPC Workshop, Houston, Texas, March 2008. Invited talk.

- J. Mellor-Crummey. Sampling-based Measurement and Analysis of Application Performance, SIAM Conference on Parallel Processing, Atlanta, Georgia, March 2008. Invited talk.

- J. Mellor-Crummey. Towards Sampling-based Performance Tools for Tuning at the Petascale, Teragrid Petascale Workshop, Las Vegas, Nevada, May 2008. Invited talk.

- J. Mellor-Crummey. HPCToolkit Components for Measurement, Analysis, and Presentation, CScADS Workshop on Performance Tools for Petascale Computing, Snowbird, Utah, July 2008.

- J. Mellor-Crummey. Programming Models for Scientific Computing on Leadership Computing Platforms: The Evolution of Coarray Fortran, CScADS Workshop on Workshop on Leadership-Class Machines, Petascale Applications, and Performance Strategies, Snowbird, Utah, July 2008.

- J. Mellor-Crummey. Towards Sampling-based Performance Tools for Tuning at the Petascale, CScADS Workshop on Workshop on Leadership-Class Machines, Petascale Applications, and Performance Strategies, Snowbird, Utah, July 2008.

- J. Mellor-Crummey. Programming Models for Scientific Computing on Leadership Computing Platforms: The Evolution of Coarray Fortran, CScADS Workshop on Workshop on Leadership-Class Machines, Petascale Applications, and Performance Strategies, Snowbird, Utah, July 2008.

- J. Mellor-Crummey. Towards Sampling-based Performance Tools for Tuning at the Petascale, CScADS Workshop on Workshop on Leadership-Class Machines, Petascale Applications, and Performance Strategies, Snowbird, Utah, July 2008.

- J. Mellor-Crummey. Gaining Insight into Parallel Program Performance using Sampling. Workshop on Performance Analysis of Extreme-Scale Systems and Applications, Los Alamos Computer Science Symposium, 2009. Santa Fe, NM, USA.

- J. Mellor-Crummey, HPCToolkit: Sampling-based performance tools for leadership computing. Center for Scalable Application Development Software's Workshop on Leadership-class Machines, Petascale Applications, and Performance Strategies. Tahoe City, CA, USA, July 27-30, 2009.

- J. Mellor-Crummey. A New Vision for Coarray Fortran. Workshop on Non-Traditional Programming Models for High-Performance Computing, Los Alamos Computer Science Symposium, Santa Fe, NM, USA, October 14, 2009.

- N. Tallent, Performance Measurement and Analysis of Multithreaded Programs. Center for Scalable Application Development Software's Workshop on Performance Tools for Petascale Computing. Tahoe City, CA, USA, July 20-23, 2009.

- N. Tallent. Binary analysis for measurement and attribution of program performance. In ACM SIGPLAN Conference on Programming Language Design and Implementation (Dublin, Ireland, June 15 - 21, 2009).

- N. Tallent. Effective performance measurement and analysis of multithreaded applications. In 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (Raleigh, NC, USA, February 14 - 18, 2009).

- Nathan Tallent. Using sampling to understand parallel program performance. 5th Parallel Tools Workshop (Dresden, Germany, September 2627, 2011).

- J. Sandoval. Dynamic Compilation for CCA Applications. CCA Forum Meeting. Rockville, MD, April 23, 2009.