Performance Engineering Research Institute
SciDAC-2 Enabling Technologies Institute
Final Report

Investigators

Lead investigators from the original PERI consortium:

Argonne National Laboratory	Paul Hovland and Boyana Norris
Lawrence Berkeley National Laboratory	David Bailey and Kathy Yelick
Lawrence Livermore National Laboratory	Bronis de Supinski and Dan Quinlan
Oak Ridge National Laboratory	Pat Worley, Jeff Vetter and Phil Roth
Rice University	John Mellor-Crummey
University of California at San Diego	Allan Snavely
University of Maryland	Jeff Hollingsworth
University of North Carolina	Rob Fowler
University of Southern California	Bob Lucas and Jacqueline Chame
University of Tennessee at Knoxville	Jack Dongarra and Shirley Moore
University of Utah	Mary Hall

Other investigators who have received funding via PERI in FY2010:

North Carolina State University	G. Mahinthakumar

I. Introduction

This document reports on work performed by the Performance Engineering Research Institute (PERI), an Enabling Technologies Institute of the SciDAC-2 program of the Department of Energy's Office of Science (DOE-SC). Since Utah PI Mary Hall led the autotuning effort, this report focuses on the Utah contributions to PERI.

Enhancing the performance of SciDAC applications on petascale systems has high priority within DOE SC. As we look to the future, achieving expected levels of performance on high-end computing (HEC) systems is growing ever more challenging due to enormous scale, increasing architectural complexity, and increasing application complexity. To address these challenges, PERI has implemented a unified, tripartite research plan encompassing: (1) performance modeling and prediction; (2) automatic performance tuning; and (3) performance engineering of high profile applications. The PERI performance modeling and prediction activity is developing and refining performance models, significantly reducing the cost of collecting the data upon which the models are based, and increasing model fidelity, speed and generality. Our primary research activity is automatic tuning *(autotuning)* of scientific software. This activity is spurred by the strong user preference for automatic tools and is based on previous successful activities such as ATLAS, which has automatically tuned components of the LAPACK linear algebra library, and other recent work on autotuning domain-specific libraries. Our third major component is application engagement, to which we are devoting approximately 30% of our effort to work directly with SciDAC-2 applications. This last activity not only helps DOE scientists meet their near-term performance goals, but also helps keep PERI research focused on the real challenges facing DOE computational scientists as they enter the Petascale Era.

During the project, the original PERI consortium of ten institutions has been augmented with additional researchers, either via subcontracts or because the researchers changed institutions and their funding has followed. Inasmuch as these participants make significant contributions to the

overall PERI goals and are now being funded directly or indirectly through PERI, we have included them in this report.

Accomplishments

As mentioned above, PERI has three primary activities: (a) application performance modeling, (2) automatic performance tuning, and (3) application engagement. A brief overview of accomplishments in each of these areas is presented below. This is followed by a brief discussion of other activities, such as management and outreach to others in the performance community. This report is a summary of the overall progress of the PERI effort towards autotuning. PERI project reports cover the remaining activities.

II. Automatic Performance Tuning

In the area of automatic performance tuning, PERI researchers are developing whole program analyses that facilitate optimizations spanning multiple kernels, files, and procedure boundaries. This work utilizes autotuning techniques on significantly larger scales than individual kernels, and is targeted at full-scale SciDAC applications. The principal research focus involves scaling and generalizing the techniques developed for kernel-level optimization and in leveraging work elsewhere within PERI to identify opportunities for improvement and then generate viable search strategies. As an outgrowth of collaborations with SciDAC application teams, we are defining additional analysis and transformation support required to meet the needs of SciDAC application developers. Architectural targets include utilizing SIMD compute engines such as SSE-3, prefetch into cache, and managing multiple cores. In addition, PERI personnel are developing transformations for specific application classes such as stencil computations, and specialization for known problem sizes. A number of transformation mechanisms are being explored to simplify the development of application-specific transformations. This work will result in a library of transformations that are easy to specify and compose to create custom transformations. Mary Hall of Utah is leading the PERI automatic performance tuning activity.

Autotuning Tool Integration

Within PERI, several different research groups are developing autotuning *tools* to address the challenges of automatic performance tuning. These projects, many of which have benefitted from years of DOE investment, have complementary strengths and can, therefore, be brought together to develop an integrated autotuning *system*. Towards that end, PERI researchers are working to develop a common framework to allow autotuning tools to share information and facilitate composition into the most appropriate set of tools for a particular application. Through common application programming interfaces (APIs), they are creating an autotuning system that brings together the best capabilities of each of these tools. They anticipate that such a strategy will also engage the broader community of tool developers beyond PERI researchers.

PERI researchers have focused their development of interfaces on two portions of the autotuning process. Any compiler-based approach will apply code transformations to rewrite application code from its original form to a form that more effectively exploits architectural features such as registers, caches, SIMD compute engines, and multiple cores. Commonly used code transformations include loop unrolling, blocking for cache, and software pipelining. Thus, researchers are designing a *transformation API* that is input to the Transformation box in Figure 1. This API provides a *transformation recipe* that describes how to transform original source into an optimized source representation.

By accepting a common transformation recipe, the autotuning system permits code transformation strategies derived by PERI compilers and tools (or users) to be implemented using any transformation and code generation tool, such as CHiLL (USC/ISI, Utah and Argonne), the ROSE LoopProcessor (LLNL), and POET (UTSA). The API supports the specification of unbound

transformation parameters that are then tuned using search algorithms. The initial API includes a naming convention for specifying language constructs in source code and code transformations available in CHiLL and ROSE/POET. Researchers have published two papers describing transformation recipes [Hall2009][Rudy2010]. The first paper, a collaboration between University of Utah, USC/ISI and Argonne researchers, describes an API that attempts to generalize concepts derived from the PERI team into a common recipe that can be the input/output of different PERI tools. The second paper, produced in collaboration between University of Utah and USC/ISI, raises the level of abstraction for this framework and offers a programming language interface for developing libraries of optimization strategies. Such optimization libraries can be programmed by compiler experts, and then made available to application developers, thus providing a high-level interface to the compiler code generation and transformation and the auto-tuning framework.

A *search API* provides input into the empirical optimization process by running experiments on actual hardware to determine the best optimized implementation. The search API allows the autotuning tools to exchange information about their available tuning options and constraints on the search space, and to plug-in different search algorithms. The common framework will support both autotuning using training runs (and re-compilation) along with continuous optimization during production runs. The search API, led by the UMD team, permits autotuning systems to exchange information about their available tuning options, constraints on the search space, and to plug-in different search algorithms. The common framework supports both automatic tuning using training runs along with continuous optimization during production runs.

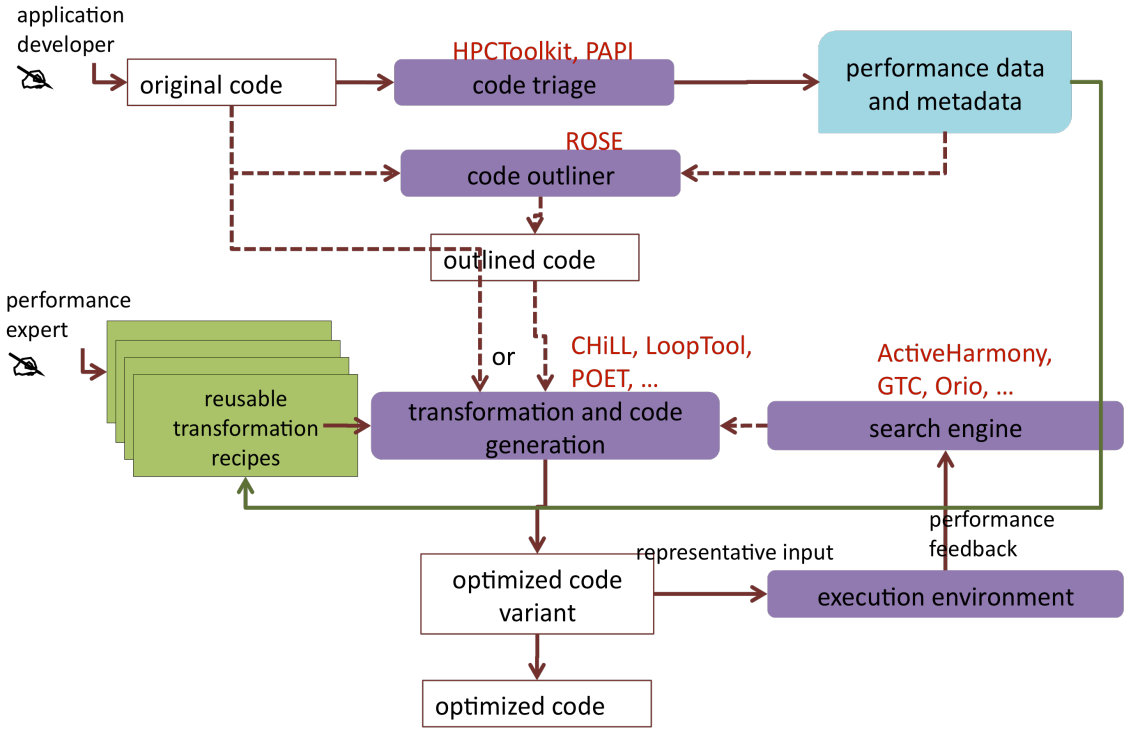

Figure 1: PERI automatic tuning workflow.

PERI work in autotuning has focused on the integration of several PERI tools. Researchers at UTK have integrated autotuning search (described below) with the ROSE LoopProcessor (LLNL) and the POET code generator (UTSA). Similarly, an initial integration of UMD's Active Harmony system and the CHiLL transformation framework developed and supported by researchers at USC/ISI, Utah and ANL is providing experience in how to integrate these separate tools effec-

tively into an autotuning system. Through the APIs and substantial coordination activities, PERI researchers are now focused on more extensive integration of end-to-end tools applied to complete applications. At the mid-term review, researchers described how this integrated set of tools was applied to SMG2000. Later in the project, they applied these tools to PFLOTRAN as part of the Tiger Team activity [Chame2011]. Inspired by this effort, a University of Utah Masters project looked at specialization and autotuning of PETSc in the context of three applications, PFLOTRAN and two others [Ramalingam2012a] [Ramalingam2012b]. Most of the remainder of this section describes work on PFLOTRAN, PETSc, and the research activities being carried out by the PERI groups, and where relevant, integration activities between the tools.

Autotuning of PFLOTRAN

PFLOTRAN is a DOE application developed at LANL that models multiscale-multiphase-multicomponent subsurface reactive flows. PFLOTRAN uses the PETSc library as the basis of its parallel framework. As shown in Figure 8, PERI researchers integrated tools that included the work of several PERI institutions to perform auto-tuning of key computations in PFLOTRAN. A *code triage* phase examines the original PFLOTRAN code to identify the key computations. We use HPCToolkit, which in turn uses PAPI, to collect performance monitoring information and map it back to the application code structures.

Using HPC Toolkit (Rice) and the Cray PAT tool, researchers at North Carolina State, Rice and ORNL identified three main computations in PFLOTRAN as candidates for optimization: a PETSc routine that computes a matrix-vector multiplication (MatMul_SeqBAIJ_N); a PETSc function that solves the system A x = b, given a factored matrix A, (MatSolve_SeqBAIJ_N); and a routine that calculates the contribution of aqueous equilibrium complexity to the residual and Jacobian functions for Newton-Raphson (RTOTAL). These two PETSc functions comprise 17% of execution time, and 6-7% of the computation is spent in a single loop nest computation of RTOTAL; all functions achieve only between 4 and 5% of peak on a node on Jaguar.

In a collaboration between USC/ISI, Utah and UMD, PERI researchers performed autotuning of MatSolve_SeqBAIJ_N, which performs a forward and a backward solve, and uses a blocked compressed row representation for matrix A. The block size N is a parameter, but instrumenttion data shows that during production runs the block sizes are small numbers, up to 15. As previous work by the Utah, USC and ANL has shown [Shin2010], the performance of matrix computations with small matrix sizes is very sensitive to optimization parameters such as loop unroll factors and tile sizes. When combined with specialization, a compiler technique for generating highly optimized code for known problem sizes, we can achieve better performance than most high performance libraries. The auto-tuning team used CHiLL to generate specialized code versions for a particular block size, and ActiveHarmony to search for the best performing version. Our experiment used four different compilers available on Jaguar, and evaluated performance as a function of unroll factors for the inner loops. PERI researchers compared results using Active Harmony and an exhaustive search of a search space consisting of more than 1100 points. The results are shown in Table 1. The original performance varies significantly by compiler, but the best speedup and best overall performance was obtained with the Pathscale compiler, demonstrating a 1.8x performance gain with Active Harmony and a 1.9x performance gain using exhaustive search. UMD and Utah are currently working on replacing this code in the full PFLOTRAN application. USC/ISI is currently working on the performance optimization of the backward solve targeting the Cray XT5 at ORNL (Jaguar).

Com-piler	Original	Active Harmony			Exhaustive		
	Time	Time	(u1,u2)	Speedup	Time	(u1,u2)	Speed up
path-scale	0.58	0.32	(3,11)	1.81	0.30	(3,15)	1.93
gnu	0.71	0.47	(5,13)	1.51	0.46	(5,7)	1.54
pgi	0.90	0.53	(5,3)	1.70	0.53	(5,3)	1.70
cray	1.13	0.70	(15,5)	1.61	0.69	(15,15)	1.63

Table 1. Auto-tuning results for forward solve from triangular solve library, using CHiLL and Active Harmony.

In a collaboration between Argonne, Utah and UMD, we performed auto-tuning for the PETSc routine MatMul_SeqBAIJ_N. Like the previous example, the code was specialized for block sizes of N=15 or a multiple of 15 (15x105,15x90,15x75, 15x60). Performance gains were up to 1.5X on a single node of Jaguar using the PGI compiler.

In addition to these two PETSc kernels, we also optimized the RTOTAL function from PFLOTRAN. Early pathfinding and hand optimization by ORNL researchers revealed a collection of optimizations that would improve the performance of the loop from this code in which the application spent 6% of its time. The loop consisted of inner loop nests with a variable *ncomp* used to establish the number of loop iterations. By analyzing instrumentation data, we discovered that the value of *ncomp* was usually very small, between 2 and 4. By specializing the code for specific values of ncomp and aggressively unrolling the inner loop nests, we can achieve speed-ups ranging from 1.32X to 1.52X using CHiLL, or up to 1.8X with additional manual optimizations to remove recurrences between accumulations. These results are summarized in Table 2.

Ncomp	Original	Chill		Hand Tuned	
	Time	Time	Speedup	Time	Speedup
1	0.09	0.06	1.52	0.05	1.80
2	0.13	0.096	1.32	0.087	1.46
3	0.18	0.12	1.49	0.11	1.45
4	0.21	0.16	1.32	0.15	1.45

Table 2. Results of optimizing RTOTAL function in PFLOTRAN.

At SciDAC 2011, a poster and short paper reported on overall performance improvement for PFLOTRAN [Chame 2011]. Beyond the kernel optimization, initial profiling studies on Cray XT5 indicated that I/O is a major bottleneck for scaling PFLOTRAN beyond 32K cores. This is primarily because parallel I/O libraries such as HDF5 that rely on MPI-IO do not scale well beyond 10K processor cores, especially on parallel file systems (like Lustre) with single point of resource contention. PERI's I/O optimization efforts led to a two-phase I/O approach at the application level where a set of designated processes participate in the I/O process by splitting the I/O operation into a communication phase and a disk I/O phase. The designated I/O processes are created by splitting the MPI global communicator into multiple sub-communicators. The root process in each sub-communicator is responsible for performing the I/O operations for the entire group and then distributing the data to rest of the group. This approach resulted in over 25X speedup in HDF I/O read performance and 3X speedup in write performance for PFLOTRAN at over 100K processor cores on the ORNL's Cray XT5 systems [Sripathi 2009][Sripathi 2010].

Final PFLOTRAN results. After autotuning, each of these routines was sped up by nearly a factor of two. Additional I/O tuning described above resulted in a final overall 40-fold speedup of the initialization phase, 4-fold improvement in the write stage, and a 5-fold improvement of total simulation time on 90,000 cores.

Autotuning and Specialization of PETSc

Libraries such as PETSc are written in a very general way to anticipate a wide variety of ways in which they may be used. This generality may lead to extensive control flow tests or other overheads, and reduces optimization opportunities. Through instrumentation, we may be able to identify common use cases within a specific application, and improve optimization effectiveness when such information is available. For example, selecting unroll factors and tile sizes for loop nests benefits from information about the iteration count and memory accesses within the nest.

During the optimization of PFLOTRAN above, instead of writing manually optimized versions of PETSc library calls and testing for different unroll/tiling factors, the kernel along with its known parameters were provided as inputs to CHiLL. CHiLL was used to generate different code variants according to the parameter values and transformation factors. A heuristic based search was then performed by Active Harmony to find the best performing variant for each permutation of values a set of parameters/variables can possess. The experiment was performed swiftly and the overall applications performance was improved by 5%.

Writing specialized code is a technique often used by library developers to optimize applications. However, manually-written code has several disadvantages when compared to our framework.

- The library developer will not possess the values of parameters along with their frequencies at design time. Hence, he would be able to write specialized functions only for specific values.
- It gets a little difficult for the developer to reason about the best implementation when the number of variables/parameter along with each of their possible values is more than a few.
- It is not feasible to expect the application programmer to write specialized code.
- Performance of implementations vary according to the architecture.

PETSc alone has 29 functions which have been specialized, amounting to a total of 242 manually-written functions. For the last few months we have been investigating how to combine CHiLL with PETSC such that applications can use the CHiLL framework to generate these specialized versions to reduce the amount of code that is provided by the library and, using auto-tuning and specialization, derive more highly optimized versions of the library functions. This code generation could be deferred until the build of the application so that the code can be specialized for the application and execution context. Such an integration would allow users to study the hotspots in the PETSC library and extract frequent values if possible. The best performing code would then be automatically generated and included in the PETSC library.

Figure 2 shows results from a study using CHiLL to specialize PETSc code for three large-scale applications: PFLOTRAN (as previously described), the Uintah Problem Solving Framework and UNIC, a 3D unstructured deterministic neutron transport code. This work demonstrated significant performance improvements of more than 1.8X on the library functions and overall gains of 9 to 24% on the overall applications. A full report of this experiment and methodology can be found elsewhere [Ramalingam2012a][Ramalingam2012b].

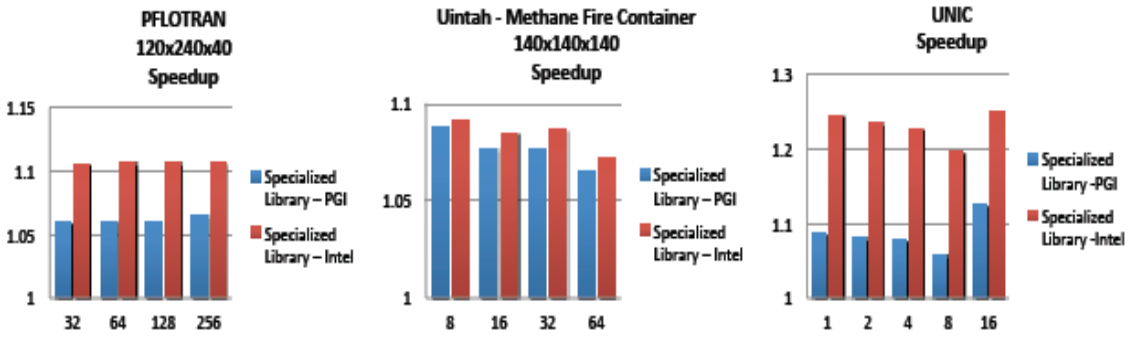

Figure 2. Impact of PETSc specialization on application performance.

Autotuning Technology Development

PERI researchers at University of Maryland enhanced their Active Harmony framework to permit new code to be compiled during program execution and loaded into a running application (*on-line performance tuning).* As we explore a broader search space of transformation strategies in PERI, often exponential, it is not feasible to generate all possible code variants off-line and then select among these generated versions of the code. By dynamically generating the code variants, we can generate the code in the execution context of the running application. The tuned variants are implemented with wrapper functions that query Active Harmony to receive the status of code generation. Once the new code is ready, the old code is transparently replaced with new code using the dlopen-dlsym mechanism. Each node running the application keeps track of the best code-variant it has seen thus far in the tuning process. When the code-server fails to deliver new versions on time, the nodes continue their execution with the best version that they have discovered until that point in the tuning process. The decision about what code-variants to generate and evaluate at each iteration is made completely by the centralized tuning server. The UMD team also investigated the important technical question of how many nodes are required to generate all of the code variants. On a specific Lattice-Boltzmann example, we were able to achieve up to a 48% speedup in a single run by using only 1.5% additional cores for code generation.

PERI researchers at Argonne continued extending Orio, which is an annotation-based system for automated generation of multiple tuned versions of critical computations. Researchers have successfully applied Orio to both dense linear algebra kernels and block sparse matrix computations in PETSc. We have also applied it to Fortran kernels based on FLASH using the new Fortran code generation capabilities. Understanding the performance of applications prior to analysis or tuning is a labor-intensive process because of differences among architectures and performance tool interfaces. Working closely with the SciDAC Center for Technology for Advanced Scientific Component Software (TASCS) center, Argonne researchers developed a set of prototype components for managing performance experiments for arbitrary parallel applications. The goal of this effort is to simplify and automate the process of defining and executing performance experiments and subsequent performance data storage and analysis.

Utah, USC/ISI and ANL have developed CHiLL, a framework for composing high-level loop transformations designed to generate efficient code for complex loop nests. It supports an extensive set of loop transformations for perfect and imperfect loop nests, including tiling, permutation and unroll-and-jam, thus lifting the burden of generating multiple intermediate steps from compilers or optimization tools. CHiLL uses an improved version of Omega (OmegaPlus) to manipulate integer arithmetic and relies on polyhedral scanning provided by Omega's code generator. Utah, USC/ISI and UMD researchers have been working to integrate the Active Harmony system with the CHiLL framework. Users can customize the CHiLL optimizer via a process called recipes. This integrated framework was applied to the problem of auto-tuning PFLOTRAN, and the previously-described results were from the combination of CHiLL and Active Harmony. In the past year, we have performed additional experiments applying CHiLL to two SciDAC-e codes, MGDC and the QR factorization of <??>. A recent integration of CHiLL with the ROSE compiler is undergoing extensive testing in preparation for release within the next year.

PERI researchers at University of Tennessee Knoxville (UTK) have developed an auto-tuning process for parallel I/O. The process combines a mathematical model with simulation and empirical search. We use the mathematical model to generate a set of parameters that serve as the starting point of the tuning process. Each set of parameters is input to the simulation, which has been calibrated using an I/O benchmark. In addition, the UTK team has continued to develop DAG-based scheduling of tasks and automatic selection of blocking sizes for auto-tuning of dense linear software on multi-core architectures. The approach uses a pruned search methodology for the one-sided factorization algorithms.

Autotuning Software Products

The following provides information on availability of PERI autotuning software.

1. ROSE is available at http://www.roseCompiler.org, with all software and documentation including access to the SVN repository. Additional documentation provides a tutorial on how to use ROSE to connect together different PERI tools to perform whole program autotuning.
2. Active Harmony is available at http://www.dyninst.org/harmony/software/software.html
3. CHiLL and Omega Plus are available from http://www.cs.utah.edu/~chunchen/.
4. Orio is available at http://trac.mcs.anl.gov/projects/performance/wiki/Orio or http://tinyurl.com/OrioTool. CQoS database components (with TASCS) are available from http://trac.mcs.anl.gov/projects/cca/wiki/cqos.

PERI Bibliography

Books:

David H. Bailey, Robert F. Lucas and Samuel W. Williams, ed., *Performance Tuning of Scientific Applications*, Taylor and Francis, New York, manuscript now complete; to be released in late 2010.

Chapters 1, 2, 3, 4, 7,10, 11, 12, 14, 15, 16 were directly authored or co-authored by PERI-funded researchers; all chapters were at least edited by PERI-funded personnel:

1. "Introduction" by David H. Bailey
2. "Parallel Computer Architecture, by Samuel W. Williams and David H. Bailey
3. "Software Interfaces to Hardware Counters," by Shirley V. Moore, Daniel K. Terpstra, and Vincent M. Weaver.
4. "Measurement and Analysis of Parallel Program Performance using TAU and HPCToolkit," by Allen D. Malony, John Mellor-Crummey, and Sameer S. Shende.
5. "Trace-Based Tools," by Jesus Labarta (Labarta is at the University of Barcelona in Spain, but has collaborated with PERI in application analysis).
6. "Large-Scale Numerical Simulations on High-End Computational Platforms," Leonid Oliker, Jonathan Carter, Vincent Beckner, John Bell, Harvey Wasserman, Mark Adams, Stephane Ethier, and Erik Schnetter.
7. "Performance Modeling: The Convolution Approach," by David H Bailey, Allan Snavely, and Laura Carrington.
8. "Analytic Modeling for Memory Access Patterns Based on Apex-MAP," by Erich Strohmaier, Hongzhang Shan, and Khaled Ibrahim.
9. "The Roofline Model," by Samuel W. Williams.
10. "End-to-end Auto-tuning with Active Harmony," by Jeffrey K. Hollingsworth and Ananta Tiwari.
11. "Languages and Compilers for Auto-Tuning," by Mary Hall and Jacqueline Chame.
12. "Empirical Performance Tuning of Dense Linear Algebra Software," by Jack Dongarra and Shirley Moore.
13. "Auto-Tuning Memory-Intensive Kernels for Multicore." By Samuel W. Williams, Kaushik Datta, Leonid Oliker, Jonathan Carter, John Shalf, and Katherine Yelick.
14. "Flexible Tools Supporting a Scalable First-Principles MD Code," Bronis R. de Supinski, Martin Schulz, and Erik W. Draeger.
15. "The Community Climate System Model," by Patrick H. Worley.
16. "Tuning an Electronic Structure Code," by David H Bailey, Lin-Wang Wang, Hongzhang Shan, Zhengji Zhao, Juan Meza, Erich Strohmaier, and Byounghak Lee.

Technical Articles:

[Barnes2010] Bradley Barnes, Jeonifer Garren, David K. Lowenthal, Jaxk Reeves, Bronis R. de Supinski, Martin Schulz and Barry Rountree, "Using Focused Regression for Accurate Time-Constrained Scaling of Scientific Applications," *Twenty Fourth International Parallel and Distributed Processing Symposium (IPDPS 2010)*, Atlanta, GA, Apr 19-23, 2010.

[Bedard2010] Daniel Bedard, Min Yeol Lim, Robert Fowler, and Allan Porterfield, "PowerMon: Fine-grained and integrated power monitoring for commodity computer systems," in *Proceedings Southeastcon 2010*, Charlotte, NC, IEEE, March 2010.

[Bedard2009] Daniel Bedard, Allan Porterfield, Rob Fowler and Min Yeol Lim, "PowerMon 2: Fine-grained, Integrated Power Measurement," Technical Report TR-09-04, RENCI, North Carolina, October 2009.

[Boehme2010] David Boehme, Martin Schulz, Bronis R. de Supinski, Markus Geimer and Felix Wolf, "Critical Path Analysis for Large-Scale MPI Programs," poster at SC2010, New Orleans, Louisiana, Nov 13-19, 2010.

[Bui2009] V. Bui, B. Norris, and L. C. McInnes. An automated component-based performance experiment environment. *Proceedings of the 2009 Workshop on Component-Based High Performance Computing (CBHPC 2009)*, Nov. 2009.

[Chame 2011] Jacqueline Chame, Chun Chen, Mary Hall, Jeffrey K. Hollingsworth, Kumar Mahinthakumar, Gabriel Marin, Shreyas Ramalingam, Sarat Sreepathi, Vamsi Sripathi, Ananta Tiwari, "PERI Autotuning of PFLOTRAN," Journal of Physics (to appear), Proceedings of SciDAC 2011, July 2011.

[Gamblin2010] Todd Gamblin, Bronis R. de Supinski, Martin Schulz, Robert J. Fowler and Daniel A. Reed, "Clustering Performance Data Efficiently at Massive Scales," *Twenty Fourth International Conference on Supercomputing (ICS 2010)*, Tsukuba, Japan, Jun 1-4, 2010.

[Haidar2011] Haidar, A., Ltaief, H., YarKhan, A., Dongarra, J. "Analysis of Dynamically Scheduled Tile Algorithms for Dense Linear Algebra on Multicore Architectures," IEEE International Parallel and Distributed Processing Symposium (submitted), Anchorage, AK, May 16-20, 2011.

[Hall2009] M. Hall, J. Chame, C. Chen, J. Shin, G. Rudy, M. Khan, Loop Transformation Recipes for Code Generation and Auto-Tuning, *The 22nd International Workshop on Languages and Compilers for Parallel Computing*, October 8-10, 2009.

[Li2009] Dong Li, Bronis R. de Supinski, Martin Schulz, Kirk W. Cameron and Dimitrios S. Nikolopoulos, "Model-Based Hybrid MPI/OpenMP Power-Aware Computing," aposter at SC2009, Portland, Oregon, November 14–20, 2009.

[Li2010] Dong Li, Bronis R. de Supinski, Martin Schulz, Kirk Cameron and Dimitrios S. Nikolopoulos, "Power-aware MPI Task Aggregation Prediction for High-End Computing Systems," *Twenty Fourth International Parallel and Distributed Processing Symposium (IPDPS 2010)*, Atlanta, GA, Apr 19-23, 2010.

[Li2010b] Dong Li, Dimitrios S. Nikolopoulos, Kirk Cameron, Bronis R. de Supinski, and Martin Schulz, "Hybrid MPI/OpenMP Power-Aware Computing," *Twenty Fourth International Parallel and Distributed Processing Symposium (IPDPS 2010)*, Atlanta, GA, Apr 19-23, 2010.

[Liao2010] Chunhua Liao, Chunhua, Daniel J. Quinlan, Thomas Panas and Bronis R. de Supinski, "A ROSE-based OpenMP 3.0 Research Compiler Supporting Multiple Runtime Libraries," *Sixth International Workshop on OpenMP (IWOMP 2010)*, Tsukuba, Japan, Jun 14-16, 2010.

[Lim2010] Min Yeol Lim, Allan Porterfield, and Robert Fowler, "SoftPower: Fine-Grain Power Estimations Using Performance Counters," in *ACM International Symposium on High Performance Distributed Computing (HPDC)*, Chicago, IL, ACM, July 2010. Best short paper award.

[Mandal2010] Anirban Mandal, Rob Fowler, and Allan Porterfield, "Modeling memory concurrency for multi-socket multi-core systems," in *Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS2010)*, pg. 56–75, White Plains, NY, IEEE, March 2010.

[Mandal2010b] Anirban Mandal, Min Yeol Lim, Allan Porterfield, and Robert Fowler, "Implications for Applications and Compilers of Multi-core Memory Concurrency," *International Workshop on Languages and Compilers for Parallel Computing (LCPC2010)*, Houston, TX, Oct. 2010.

[Mueller2010] Frank Mueller, Xing Wu, Martin Schulz, Todd Gamblin and Bronis R. de Supinski, "ScalaTrace: Tracing, Analysis and Modeling of HPC Codes at Scale," Para 2010: State of the Art in Scientific and Parallel Computing, Reykjavík, Iceland, Jun 6-9, 2010.

[Mills2009a] Richard Tran Mills, Vamsi Sripathi, G. (Kumar) Mahinthakumar, Glenn E. Hammond, Peter C. Lichtner, Barry F. Smith, Experiences and Challenges Scaling PFLOTRAN, a PETSc-based Code for Subsurface Reactive Flow Simulations, Towards the Petascale on Cray XT Systems, *Cray Users Group Meeting*, May 2009, Atlanta, GA, 14 pages.

[Mills2009b] Mills, RT, GE Hammond, PC Lichtner, V. Sripathi, G (Kumar) Mahinthakumar, and BF Smith, (2009). Modeling Subsurface Reactive Flows Using Leadership-Class Computing, Journal of Physics, 180 (2009) 012062 doi:10.1088/1742-6596/180/1/012062.

[Mills2010] Mills, R.T., V. Sripathi, G. (Kumar) Mahinthakumar, G. E. Hammond, P. C. Lichtner, Barry F. Smith, (2010). Engineering PFLOTRAN for Scalable Performance on Cray XT and IBM BlueGene Architectures, poster presentation, SciDAC 2010.

[Muszala2009] S. Muszala, J. Amundson, L. C. McInnes, and B. Norris. Two-tiered component design and performance analysis of Synergia2 accelerator simulations. *Proceedings of the 2009 Workshop on Component-Based High Performance Computing (CBHPC 2009)*, Nov. 2009.

[Olschanowsky2010] Catherine Mills Olschanowsky, Tajana Rosing, Allan Snavely, Laura Carrington, Mustafa M. Tikir and Michael Laurenzano, "Fine-grained Energy Consumption Characterization and Modeling," *Proceedings of the 2010 DoD Users Group Conference*, to appear, 2010.

[Narayanan2010] S. H. K. Narayanan, B. Norris, and P. D. Hovland. Generating performance bounds from source code. Proceedings of the First International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI 2010), 2010.

[Pearce2010] Olga Pearce, Todd Gamblin, Martin Schulz, Bronis R. de Supinski and Nancy Amato, "Load Balance: Correlating Application-Independent Measurements with Application-Semantic Computational Models," aposter at SC2010, New Orleans, Louisiana, Nov 13-19, 2010.

[Porterfield2010] Allan Porterfield, Rob Fowler, Min Yeol Lim, "RCRTool: Design Document; Version 0.1," Technical Report TR-10-01, RENCI, North Carolina, February 2010.

[Preissl2010] Robert, Robert, Bronis R. de Supinski, Martin Schulz, Daniel J. Quinlan, Dieter Kranzlmueller and Thomas Panas, "Exploitation of Dynamic Communication Patterns through Static Analysis," *Proceedings of 2010 International Conference on Parallel Processing (ICPP-10)*, San Diego, CA, Sep 13-16, 2010.

[Preissl2010b] Robert Preissl, Martin Schulz, Dieter Kranzlmueller, Bronis R. de Supinski and Daniel J. Quinlan, "Transforming MPI Source Code Based on Communication Patterns," Future Generation Computer Systems, vol. 26, No. 1, Jan 2010, pg. 147-154.

[Ramakrishnan20009] L. Ramakrishnan, D. Nurmi, A. Mandal, C. Koelbel, D. Gannon, T. M. Huang, Y. S. Kee, G. Obertelli, K. Thyagaraja, R. Wolski, A. Yarkhan, D. Zagorodnov, "VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance", in *Proceedings of the IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC) '09*, Portland, OR, Nov 2009.

[Ramalingam2012a] S. Ramalingam, M. Hall, and C. Chen; "Improving High-Performance Sparse Libraries Using Compiler-Assisted Specialization: A PETSc Case Study," *2012 IEEE 26th*

International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), May 2012.

[Ramalingam2012b] S. Ramalingam, “"Improving High-Performance Sparse Libraries Using Compiler-Assisted Specialization: A PETSc (Portable Extensible Toolkit for Scientific Computation) Case Study," May 2012.

[Rudy2010] G. Rudy, M. Hall, C. Chen, J. Chame, M. Khan, “A Programming Language Interface to Describe Transformations and Code Generation,” *The 23rd International Workshop on Languages and Compilers for Parallel Computing*, October, 2010.

[Sarjar2009] V. Sarkar, W. Harrod and A. E. Snavely, “Software challenges in extreme scale systems,” *Journal of Physics: Conference Series*, vol.180, 012045, 2009.

[Schulz2009] Schulz, Martin, Andrew W. Cook, William H. Cabot, Bronis R. de Supinski and William D. Krauss, "On the Performance of the Miranda CFD Code on Multicore Architectures," *Twenty First International Conference on Parallel Computational Fluid Dynamics (ParallelCFD 2009)*, Moffett Field, California, May 18 -22, 2009.

[Shin2010] Speeding up Nek5000 with Autotuning and Specialization. Jaewook Shin, Mary Hall, Jacqueline Chame, Chun Chen, Paul Fisher and Paul Hovland. *The 24th International Conference in Supercomputing (ICS 2010)*, June 2010.

[Singh2010] Karan Singh, Matthew Curtis-Maury, Sally A. McKee, Filip Blagojevic, Dimitris S. Nikolopoulos, Bronis R. de Supinski and Martin Schulz, “Comparing Scalability Prediction Strategies on an SMP of CMPs,” Euro-Par 2010, Naples, Italy, Aug 31-Sep 3, 2010.

[Sripathi 2009] Vamsi Sripathi, Glenn E. Hammond, G. (Kumar) Mahinthakumar, Richard T. Mills, Patrick H. Worley and Peter C. Lichtner (2009). Performance Analysis and Optimization of Parallel I/O in a large scale groundwater application on the Cray XT5, Poster Presentation, Supercomputing 2009, Portland, Oregon, Nov 12-16.

[Sripathi2010] Sripathi, V., Performance Analysis and Optimization of Parallel I/O in a large scale Groundwater Application on Petascale Architectures, MS Thesis, North Carolina State University, August 2010.

[Tallent2010] Nathan R. Tallent, John M. Mellor-Crummey, Alan Porterfield, “Analyzing Lock Contention in Multithreaded Applications,” *Proceedings of PPoPP 2010*, January 2010.

[Tilson2010] Jeffrey L Tilson, Gloria Rendon, Eric Jakobsson, “Using high performance computing and domain-based functional annotation of proteins to enhance discovery of novel proteins, identify functional homology, and characterize phylogenetic relatedness,” Technical Report TR-10-02, RENCI, North Carolina, June 2010.

[You2011] You, H., Li, Z., Liu, Q, Moore, S. "An Auto-tuning I/O Framework for Lustre File Systems," IEEE International Parallel and Distributed Processing Symposium (submitted), Anchorage, AK, May 16-20, 2011.