Advanced Scientific Computing Research
Final Project Report

A Cross-Platform Infrastructure for Scalable Runtime Application Performance Analysis

Jack Dongarra* and Shirley Moore, University of Tennessee
Jeffrey Hollingsworth, University of Maryland
Bart Miller, University of Wisconsin

Summary

The purpose of this project was to build an extensible cross-platform infrastructure to facilitate the development of accurate and portable performance analysis tools for current and future high performance computing (HPC) architectures. Major accomplishments include tools and techniques for multidimensional performance analysis, as well as improved support for dynamic performance monitoring of multithreaded and multiprocess applications.

1.0 Introduction

Previous performance tool development has been limited by the burden of having to re-write a platform-dependent low-level substrate for each architecture/operating system pair in order to obtain the necessary performance data from the system. Manual interpretation of performance data is not scalable for large-scale long-running applications. The infrastructure developed by this project provides a foundation for building portable and scalable performance analysis tools, with the end goal being to provide application developers with the information they need to analyze, understand, and tune the performance of terascale applications on HPC architectures.

The backend portion of the infrastructure provides runtime instrumentation capability and access to hardware performance counters, with thread-safety for shared memory environments and a communication substrate to support instrumentation of multiprocess and distributed programs. Front end interfaces provides tool developers with a well-defined, platform-independent set of calls for requesting performance data. End-user tools have been developed that demonstrate runtime data collection, on-line and off-line analysis of performance data, and multidimensional performance analysis.

The infrastructure is based on two underlying performance instrumentation technologies. These technologies are the PAPI cross-platform library interface to hardware performance counters and the cross-platform Dyninst library interface for runtime modification of executable images. The Paradyn and KOJAK projects have made use of this infrastructure to build performance measurement and analysis tools that scale to long-running programs on large parallel and distributed systems and that automate much of the search for performance bottlenecks.

2.0 Hardware Counter Interface Enhancements

The PAPI library provides a portable interface to hardware performance counters available on most modern microprocessors [Browne et al., 2000]. Counters are usually available that provide counts of relevant cache and memory events, such as load and store counts and cache and TLB misses at various levels of the memory hierarchy. The PAPI effort has defined a

* 865-974-8295, dongarra@cs.utk.edu

standard set of performance metrics for the memory hierarchy, and the implementation maps these standard events to native events on a given platform. PAPI is available for most high performance computing platforms. PAPI eases the burden on developers of end-user performance analysis tools by handling the low-level details of access to hardware performance counters and providing a common interface across platforms. During this project, PAPI has been ported to a number of high-end computing systems, including Red Storm, Blue Gene/L, and Cray X1.

This project has extended the set of PAPI standard events to include additional metrics related to cache and memory performance and cache coherence events. In addition, support for accessing native events through the PAPI interface has been improved, allowing easy-to-use mnemonic names to be used instead of numeric codes. Support for using configure to automatically generate the makefiles for each platform has been added so that installation is now easier. A directory containing command-line utilities for querying the available standard and native events and for choosing a set of compatible events is now created as part of the installation. PAPI version 3 was completely rewritten to streamline the interface so that measurement overheads are much lower. Version 3 has improved support for threaded applications, and threading issues and bugs on several platforms have been resolved. The high-level interface is now thread-safe. Support was also added for interrupt on overflow for multiple events.

The PAPI substrate for the Itanium 2 includes platform-support support for profiling routines that make use of the EARs to record information about data addresses. Additional work on an end-user tool for data-centric hardware counter measurement is described in the next section.

3.0 Data Centric Measurement

During the course of this grant, we developed cacheScope, a Dyninst based tool that uses hardware counters to provide cache miss statistics on a per data structure basis. The tool uses a sampling based approach that keeps the overhead low. Cache information is provided both in terms of the share of misses due to each data structure, and the number of cycles lost due to misses.

In order to track dynamic memory allocations, calls to functions such as malloc must be replaced with equivalent calls supplied by Cache Scope. The instrumentation code uses perfmon [perfmon] to set the Itanium 2 hardware performance monitors to count L1 data cache read misses, L1 data cache reads, L2 cache misses, and Data EAR events. The Data EAR is set to record information about L1 data cache load misses and floating point loads.

For purposes of keeping statistics, memory objects are grouped into equivalence classes, which we refer to as *stat buckets*. Each global or static variable in the program is assigned its own stat bucket. When a block of memory is dynamically allocated, a bucket name is either automatically generated or is supplied by the user, as described below; this name identifies the bucket to which the block is assigned. Different blocks may have the same bucket name, so that multiple blocks are assigned to a single bucket. This is useful when a group of blocks are part of the same data structure, as in a tree or linked list. Automatically assigned names are generated based on the names of the top three functions on the call stack above the memory allocation function that allocated the object.

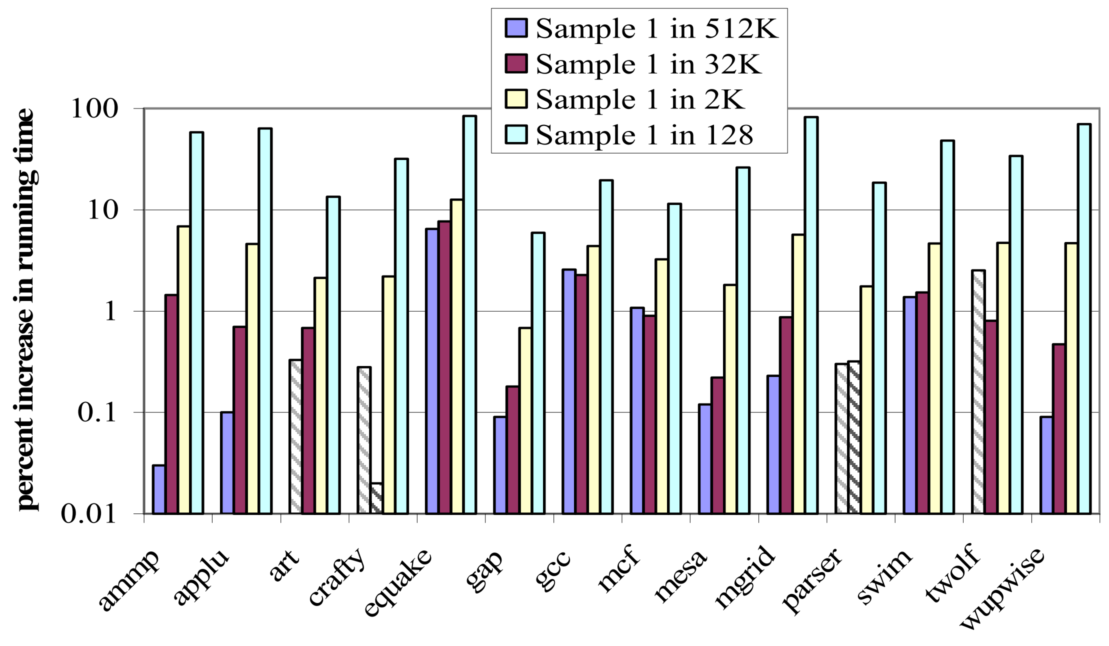

Figure 1: Overhead of Data Centric Approaches.

Figure 1 shows the overhead of sampling data cache information using our approach. The groups of bars show the overhead as the sampling rate is varied. The results show that if the sampling rate is no more than 1 in 32K misses, that the overhead of our approach is generally less than 1%, and always less than 10%.

Figure 2 shows the results of running cacheScope on the Equake program from the SPEC CPU2000 benchmark. The left column shows the data structure, and the next column shows the number of cycles of latency due to that data structure (in billion of cycles). The final column shows the average number of cycles per miss. Using this data, we were able to tune the program, and decreases L1 cache misses in the application by 57%, L2 cache misses by 30%, and running time by 10%.

The cacheScope tool is available for download from the project web page at http://www.dyninst.org/cachescope/.

Stat Bucket	Latency		
	Bcycles	%	Per Event
heap_K_2	23.67	35.4%	118.5
heap_disp_3	18.41	27.5%	10.7
heap_K_3	7.49	11.2%	5.4
heap_disp_2	3.47	5.2%	22.8
Exc	2.13	3.2%	5.5
heap_M_2	1.53	2.3%	15.2
heap_C_2	1.49	2.2%	14.7
<stack>	1.12	1.7%	5.0
heap_M_1	0.95	1.4%	23.9
heap_K_1	0.93	1.4%	80.0

Figure 2: Data Structure Statistics for Equake

4.0 On-line Automated Performance Analysis

Paradyn is a performance measurement tool for parallel programs that automates much of the search for performance bottlenecks [Miller et al., 1995]. Paradyn uses DyninstAPI [Buck and Hollingsworth, 2000] to selectively insert performance instrumentation and thus scales to long running programs and large systems. The instrumentation is controlled by the Performance Consultant which automates the runtime search for performance bottlenecks. The newest release of Paradyn has improved support for instrumenting and analyzing multithreaded and message-passing parallel applications.

Dyninst API has been extended with methods for discovering and instrumenting basic blocks and loops and functions. The newest release of Paradyn uses this extension to recognize and search loops for bottlenecks.

The latest release of Paradyn (4.2) supports MPICH and LAM MPI's, and also supports the instrumentation of OpenMP programs. It can isolate performance data to specific threads and loops. Future work includes improving the naming of loops and threads to be closer to what the Open/MP programmer specifies.

5.0 Off-line Automated Performance Analysis

The KOJAK toolkit supports performance analysis of MPI and/or OpenMP applications by automatically searching traces for execution patterns that indicate inefficient behavior [Wolf et al., 2005a]. The performance problems addressed include inefficient use of the parallel programming model and low CPU and memory performance. Figure 1 gives an overview of KOJAK's architecture and its components. The KOJAK analysis process is composed of two parts: a semi-automatic multi-level instrumentation of the user application followed by an automatic analysis of the generated performance data.

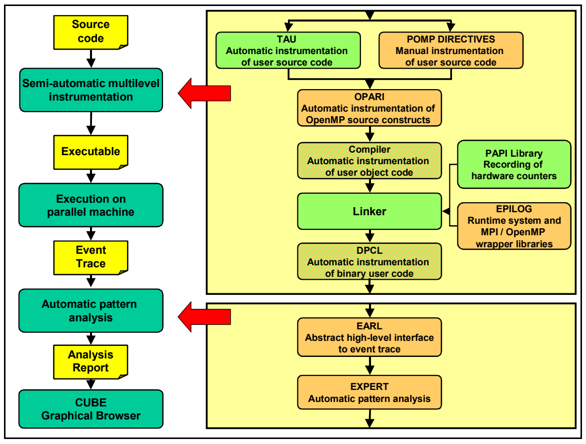
Source code
Semi-automatic multilevel instrumentation
Executable
Execution on parallel machine
Event Trace
Automatic pattern analysis
Analysis Report
CUBE Graphical Browser
TAU Automatic instrumentation of user source code
POMP DIRECTIVES Manual instrumentation of user source code
OPARI Automatic instrumentation of OpenMP source constructs
Compiler Automatic instrumentation of user object code
PAPI Library Recording of hardware counters
Linker
EPILOG Runtime system and MPI / OpenMP wrapper libraries
DPCL Automatic instrumentation of binary user code
EARL Abstract high-level interface to event trace
EXPERT Automatic pattern analysis

Figure 1: KOJAK architecture

The event traces generated by KOJAK's tracing library EPILOG capture MPI point-to-point and collective communication as well as OpenMP parallelism change, parallel constructs, and synchronization. In addition, data from hardware counters accessed using the PAPI library [Browne et al., 2000] can be recorded in the event traces. EPILOG instrumentation can be inserted using either automated source code instrumentation or the dynamic instrumentation technology described in section 3. KOJAK's EXPERT tool is an automatic trace analyzer that attempts to identify specific performance problems. EXPERT represents performance problems in the form of execution patterns that model inefficient behavior. These patterns are used during the analysis process to recognize and quantify inefficient behavior in the application. The pattern classes are organized in a specialization hierarchy, as shown in Figure 2. Recent work has taken advantage of the specialization relationships to obtain a significant speed improvement for EXPERT and to allow more compact pattern specifications [Wolf et al., 2004].

Each pattern calculates a (call path, location) matrix containing the time spent on a specific behavior in a particular (call path, location) pair, where a location is a process or thread. Thus, EXPERT maps the (performance problem, call path, location) space onto the time spent on a particular performance problem while the program was executing in a particular call path at a particular location. After the analysis has been finished, the mapping is written to a file and can be viewed using the CUBE display tool, shown in Figure 3.

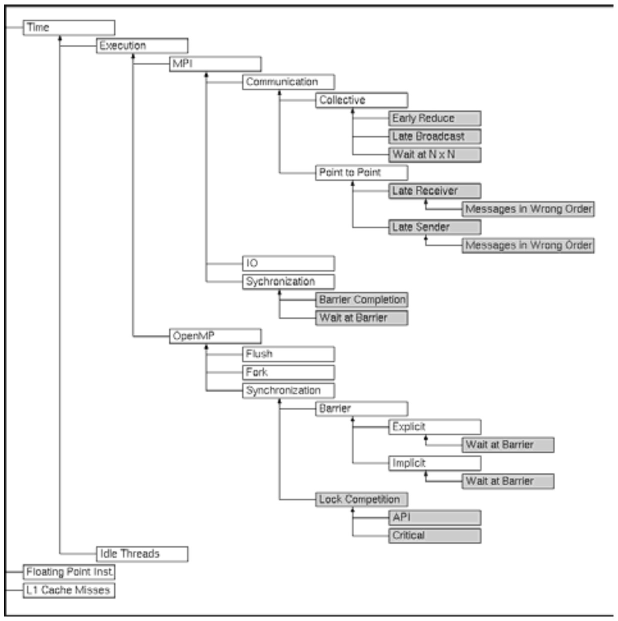

Figure 2: KOJAK pattern specialization hierarchy

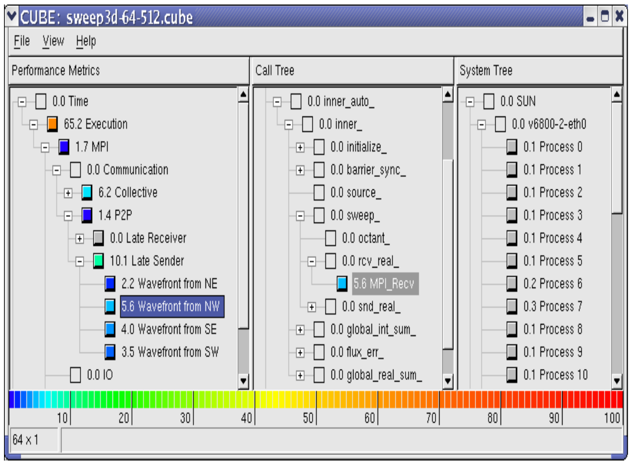

Figure 3: CUBE coupled tree browser

A performance algebra for combining multi-experiment results has also been developed that enables performance data from different executions, such as those for which different performance metrics have been collected, or those for different code versions, to be combined. The resulting derived experiment can also be viewed with CUBE [Song et al., 2004].

As part of the Performance Evaluation Research Center effort, we have used KOJAK to investigate scalability issues of the SciDAC GYRO code observed on the SGI Altix while running the B1-std benchmark [Worley et al., 2005]. We analyzed how the performance behavior of GYRO was changed after raising the number of processes from 128 to 192. Using KOJAK's performance algebra utility we have shown that the increase of consumed CPU time (linear speed up should leave the CPU time unchanged) was partly due to wait states in specific MPI_Allreduce() calls (about 1/3), partly due to increased actual communication (about 2/3), and a small amount of additional computation, as shown in Figure 4.

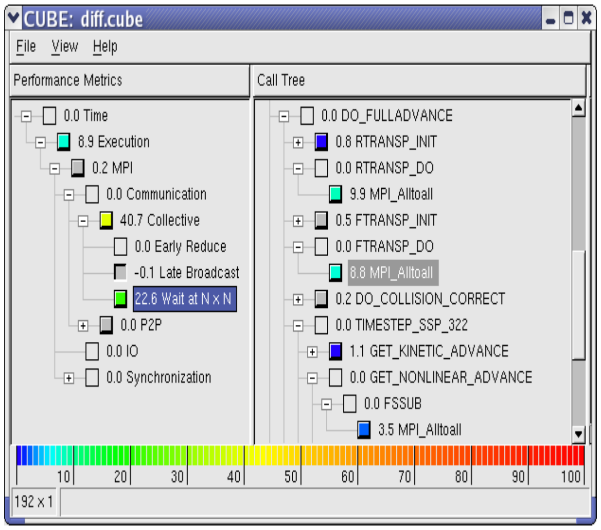

Figure 4: Change in performance behavior of GYRO when raising the number of processes on the SGI Altix from 128 to 192.

5.1 Topological Analysis

We have demonstrated that knowledge about the virtual topology, which defines logical adjacency relationships between processes, can be exploited to explain the occurrence of inefficiency patterns in terms of the parallelization strategy used in an application [Bhatia et al., 2005]. To do this we have extended KOJAK to record and process topological information in trace files. For applications that use MPI topology support, the topology is recorded automatically. For all other applications this can be done can with minimal effort by inserting two calls to a C/Fortran API into the source code. To visually map performance data onto the topology, a three-dimensional topological display has been added to CUBE. The ability to adjust various display parameters ensures scalability. Using this extension, we have shown correlations between higher-level events related to the parallel wave-front scheme used in SWEEP3D and wait states identified by our pattern analysis, as shown in Figure 5. In addition, we have visually exposed relationships between pattern occurrences and the topological characteristics of the affected processes. Finally, an extension has been implemented to automatically record the physical network topology of BlueGene/L. Scalability has been demonstrated for a run of sPPM with 1792 CPUs, as shown in Figure 6.

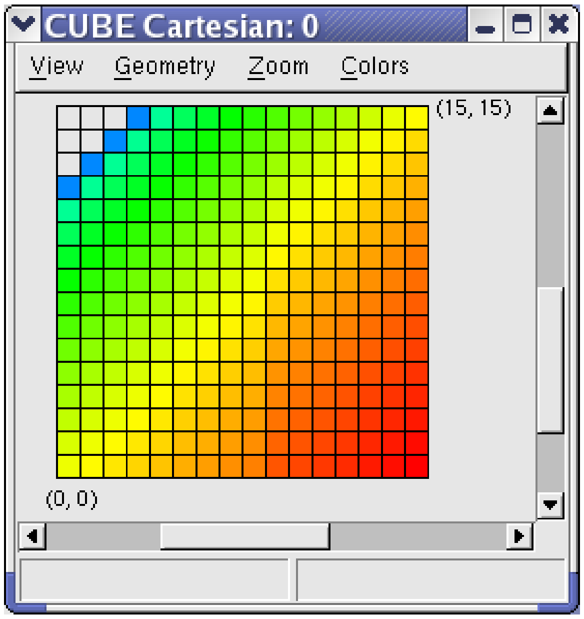

Figure 5: Distribution of wait states in SWEEP3D when pipeline is refilled from North-West.

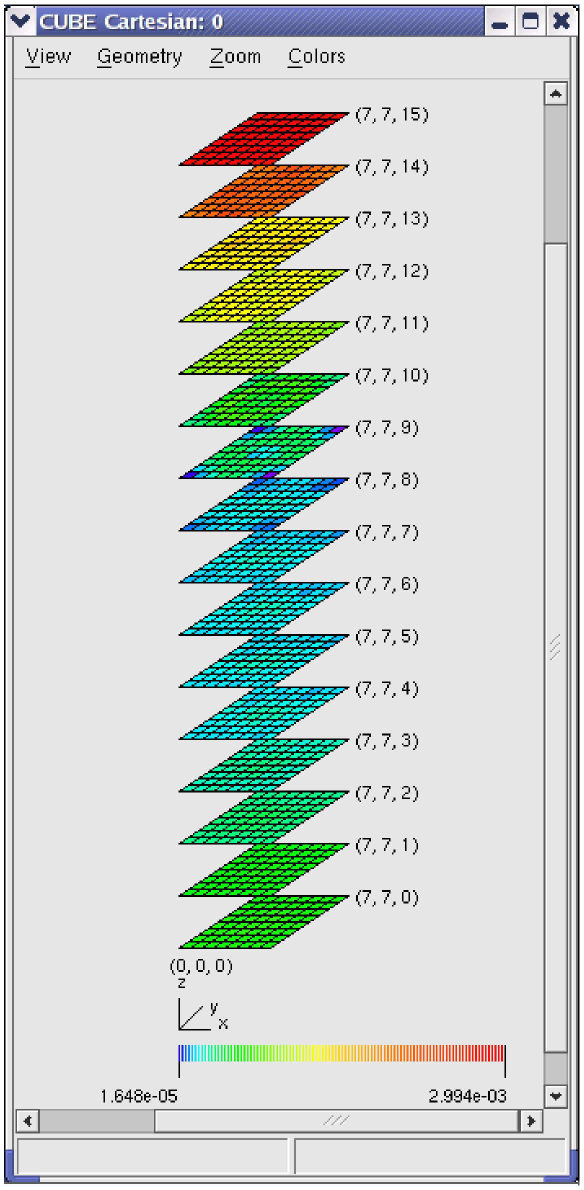

Figure 6: Distribution of wait states resulting from all-to-all operations across the torus-network topology in BG/L.

5.2 Perturbation Compensation

Tracing parallel programs to observe their performance introduces intrusion as the result of trace measurement overhead. If post-mortem trace analysis does not compensate for the overhead, the intrusion will lead to errors in the performance results. We have shown that measurement overhead can be accounted for during trace analysis and intrusion modeled and removed [Wolf et al., 2005]. Algorithms developed in earlier work have been reimplemented as part of KOJAK, allowing them to be applied in large-scale parallel programs. The ability to reduce trace measurement error has been demonstrated for a Monte-Carlo simulation based on a master/worker scheme.

5.3 Support for MPI-2

Besides other features, MPI-2 introduced a standardized interface for remote memory access (RMA). RMA allows a process to directly access a part of the memory of a remote process, without explicit participation of the remote process in the data transfer. Since all parameters for the data transfer are determined by one process, it is also called *one-sided* or *single-sided* communication. To appropriately address this programming model, we have integrated performance measurement and analysis functionality for one-sided communication into the existing KOJAK toolkit [Hermanns et al., 2005]. Special emphasis has been placed on the development of an event model that realistically represents the dynamic behavior of MPI-2 RMA operations in the event stream. It includes all synchronization methods described by the MPI-2 standard and their characteristics. Additionally, Co-Array Fortran, a vendor-specific RMA interface, that forms a subset to the model due to its simplicity, has been covered as well.

6.0 Outreach and Impact

Tools developed as part of this project have been presented at a number of workshops and tutorials, including the following:

- "Methods for Performance Engineering of Scientific Applications," tutorial, *Supercomputing 2004*, Pittsburgh, PA, November 2004.
- "Application Performance Analysis Tools for Linux Clusters", tutorial, Linux Clusters: The HPC Revolution, Austin, Texas, May 2004.
- "Performance Optimization using SvPablo and KOJAK", tutorial, Linux Clusters: The HPC Revolution, Chapel Hill, NC, April 2005.

PAPI is currently used for access to hardware counter data by a number of performance analysis tool efforts, including IPM, PerfSuite TAU, and Scalea. Integration into vendor systems is also taking place – for example, PAPI is included in the bundled software shipped with the Cray XT3, and SGI is using PAPI and Dyninst for Open SpeedShop.

Although KOJAK is a complete end-user tool, it is also an extensible infrastructure for automated performance analysis. The set of patterns on which KOJAK bases its search for performance bottlenecks is extensible, and a Python interface is supported for rapid prototyping of new patterns. KOJAK's CUBE is a generic display tool for displaying linked multiple dimensions in a performance search space.

7.0 Conclusions and Future Work

The technology developed as part of this project has been widely accepted by the high performance computing community and is already serving as the basic infrastructure for higher level performance tool development tool efforts. Continued development of the infrastructure with a focus on flexibility and scalability and its availability on cutting edge hardware will enable rapid prototyping of performance analysis tools for emerging high-end architectures.

Although the current version of PAPI supports only a single underlying substrate at a time, work is underway to vectorize PAPI so that multiple substrates can be used simultaneously. For example, this capability would allow processor performance, network performance, and power consumption information to be monitored simultaneously.

The Paradyn project has pioneered an approach of instrumenting the application to collect only that performance data relative to the performance hypotheses currently under consideration. Although Paradyn uses dynamic instrumentation, the principle of driving the data collection according to the performance questions to be answered applies to static source code instrumentation as well. As an extension of Paradyn's Metric Description Language, we plan to develop a high level Performance Description Language (PDL) that will enable the intuitive expression of performance problems in the context of the application source code. The language will have different levels of expressiveness, so that a novice user can quickly and easily specify what data should be collected for basic performance questions and the more advanced user can specify detailed expressions for more complex performance questions.

For scalability purposes, further work is needed on multi-step performance instrumentation that takes advantage of previously collected basic profile data and call-path data to restrict subsequent more detailed instrumentation to relevant portions of the execution. To make multi-step specification of instrumentation easy for the end-user, we plan to extend the KOJAK CUBE display tool to be a graphical user interface for instrumentation. The GUI would take a simple call-path profile with call-tree and baseline performance data as input. It would then show a hierarchy of potential performance problems and a corresponding annotated call-tree. Every performance tool supported would provide a "plug-in" that registers with the GUI what performance data it can collect. Clicking on a performance problem in combination with marking sub-call trees of interest would cause the appropriate instrumentation to be generated, via the formulation of PDL expressions. The translation into tool-specific instrumentation would also be guided by the plug-in.

Both Paradyn and KOJAK make use of a multi-dimensional performance search space. We plan to make use of our work on data centric measurement from this project to extend the search space for Paradyn and KOJAK with a data structure dimension. This extension will allow performance hypotheses, in the case of Paradyn, and performance patterns, in the case of KOJAK, to be expressed in terms of application data structures.

References

[Bhatia et al., 2005] Bhatia, N., Song, F., Wolf, F., Dongarra, J., Mohr, B., Moore, S.: "Automatic Experimental Analysis of Communication Patterns in Virtual Topologies". *In Proceedings of the International Conference on Parallel Processing (ICPP)*, IEEE Computer Society, Oslo, Norway, June 2005.

[Browne et al., 2000] Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P. "A Portable Programming Interface for Performance Evaluation on Modern Processors," *The International Journal of High Performance Computing Applications*, Volume 14, number 3, pp. 189-204, Fall 2000.

[Buck and Hollingsworth, 2000] Buck, B. and J. Hollingsworth, An API for Runtime Code Patching. *Journal of High Performance Computing Applications* 14(4), 2000, pp. 317-329.

[Buck and Hollingsworth, 2004] Buck, B.R. and J.K. Hollingsworth. Data Centric Cache Measurement on the Intel Itanium 2 Processor, in *SC 2004*. Pittsburgh, PA. November, 2004.

[Hermanns et al., 2005] Hermanns, M.-A. , Mohr, B., Wolf, F. "Event-based Measurement and Analysis of One-sided Communication". *In Proceedings of the European Conference on Parallel Computing (Euro-Par) (to appear)*, Springer, Lisbon, Portugal, August - September, 2005.

[Miller et al., 1995] Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam and Tia Newhall, "The Paradyn Parallel Performance Measurement Tool", *IEEE Computer* 28, 11, (November 1995): 37-46. Special issue on performance evaluation tools for parallel and distributed computer systems.

[perfmon] perfmon project web site, http://www.hpl.hp.com/research/linux/perfmon/

[Song et al., 2004] F. Song, F. Wolf, N. Bhatia, J. Dongarra, S. Moore: An Algebra for Cross-Experiment Performance Analysis. In Proceedings of the International Conference on Parallel Processing (ICPP), IEEE Computer Society, Montreal, Canada, August 2004.

[Wolf et al., 2004] Wolf, F., B. Mohr, J. Dongarra, and S. Moore. Efficient Pattern Search in Large Traces through Successive Refinement, in *European Conference on Parallel Computing (Euro-Par)*. Pisa, Italy. August-September, 2004.

[Wolf et al., 2005] Wolf, F., Malony, A., Shende, S., Morris, A. "Trace-Based Parallel Performance Overhead Compensation," *International Conference on High Performance Computing and Communications (HPCC)*, Sorrento (Naples), Italy, September, 2005. (submitted)

[Wolf et al., 2005a] Wolf, F., Mohr, B., Dongarra, J., Moore, S. "Automatic analysis of inefficiency patterns in parallel applications," *Concurrency and Computation: Practice and Experience, Special issue "Automatic Performance Analysis" (submitted)*, 2005.

[Worley et al., 2005] Worley, P., Candy, J., Carrington, L., Huck, K., Kaiser, T., Mahinthakumar, G., Malony, A., Moore, S., Reed, D., Roth, P., Shan, H., Shende, S., Snavely, A., Sreepathi, S., Wolf, F., Zhang, Y.: Performance Analysis of GYRO: A Tool Evaluation. In Proceedings of the SciDAC Conference, San Francisco, CA, 2005.

Publications, Proceedings and Reports

Mucci, P., Dongarra, J., Kufrin, R., Moore, S., Song, F., Wolf, F. **"Automating the Large-Scale Collection and Analysis of Performance,"** *In Proceedings of the 5th LCI International Conference on Linux Clusters: The HPC Revolution*, Austin, Texas, May 18-20, 2004.

Dongarra, J., Moore, S., Mucci, P., Seymour, K., You, H. **"Accurate Cache and TLB Characterization Using hardware Counters,"** *Proceedings of ICCS 2004 (to appear)*, Krakow Poland, June 6-9, 2004.

Wolf, F., Mohr, B. **"Hardware-Counter Based Automatic Performance Analysis of Parallel Programs,"** *Proc. of the Minisymposium 'Performance Analysis', Conference on Parallel Computing (PARCO)*, Elsevier, Dresden, Germany, September 3, 2003.

Dongarra, J., Malony, A., Moore, S., Mucci, P., Shende, S. **"Performance Instrumentation and Measurement for Terascale Systems,"** *ICCS 2003 Terascale Workshop*, Melbourne, Australia, June, 2003.

Dongarra, J., London, K., Moore, S., Mucci, P., Terpstra, D., You, H., Zhou, M. **"Experiences and Lessons Learned with a Portable Interface to Hardware Performance Counters,"** *PADTAD Workshop, IPDPS 2003*, Nice, France, April 26, 2003.

Moore, S., Mucci, P., Dongarra, J., Shende, S., Malony, A. **"Performance Instrumentation and Measurement for Terascale Systems,"** *Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, Volume 2723, pp. 53-62, January, 2003.

Moore, S. **"A Comparison of Counting and Sampling Modes of Using Performance Monitoring Hardware,"** *ICCS 2002*, Amsterdam, April, 2002.

Wolf, F., Malony, A., Shende, S., Morris, A. **"Trace-Based Parallel Performance Overhead Compensation,"** *International Conference on High Performance Computing and Communications (HPCC) (submitted)*, Sorrento (Naples), Italy, September, 2005.

Shende, S., Malony, A., Morris, A., Wolf, F. **"Performance Profiling Overhead Compensation for MPI Programs,"** *12th European Parallel Virtual Machine and Message Passing Interface Conference (submitted)*, Springer LNCS, September, 2005.

Moore, S., Wolf, F., Dongarra, J., Shende, S., Malony, A., Mohr, B. **"A Scalable Approach to MPI Application Performance Analysis,"** *12th European Parallel Virtual Machine and Message Passing Interface Conference (to appear),* Springer LNCS, September, 2005.

Hermanns, M.-A. , Mohr, B., Wolf, F. **"Event-based Measurement and Analysis of One-sided Communication,"** *In Proceedings of the European Conference on Parallel Computing (Euro-Par) (to appear)*, Springer, Lisbon, Portugal, August - September, 2005.

Bhatia, N., Moore, S., Wolf, F., Dongarra, J., Mohr, B. **"A Pattern-Based Approach to Automated Application Performance Analysis,"** *Workshop on Patterns in High Performance Computing*, University of Illinois at Urbana-Champaign, May, 2005.

Moore, S., Wolf, F., Dongarra, J., Mohr, B. **"Improving Time to Solution with Automated Performance Analysis,"** *Second Workshop on Productivity and Performance in High-End Computing (P-PHEC) at 11th International Symposium on High Performance Computer Architecture (HPCA-2005)*, San Francisco, February 13, 2005.

Wolf, F., Mohr, B., Dongarra, J., Moore, S. **"Automatic analysis of inefficiency patterns in parallel applications,"** *Concurrency and Computation: Practice and Experience, Special issue "Automatic Performance Analysis" (submitted)*, 2005.

Wolf, F. **"EARL - API Documentation,"** *ICL Technical Report*, ICL-UT-04-03, Oct 1, 2004.

Wolf, F., Mohr, B., Dongarra, J., Moore, S. **"Efficient Pattern Search in Large Traces through Successive Refinement,"** *Proceedings of Euro-Par 2004*, Springer-Verlag, Pisa, Italy, August 31 - Sept. 3, 2004.

Song, F., Wolf, F., Bhatia, N., Dongarra, J., Moore, S. **"An Algebra for Cross-Experiment Performance Analysis,"** *2004 International Conference on Parallel Processing (ICCP-04)*, Montreal, Quebec, Canada, August 15-18, 2004.

Song, F., Wolf, F. **"CUBE User Manual,"** *ICL Technical Report*, ICL-UT-04-01, February 2, 2004.

Wolf F., Mohr, B. **"Automatic performance analysis of hybrid MPI/OpenMP applications,"** *Journal of Systems Architecture, Special Issue 'Evolutions in parallel distributed and network-based processing'*, Clematis, A., D'Agostino, D. eds. Elsevier, 49(10-11), pp. 421-439, November, 2003.

Mohr, B., Wolf, F. **"KOJAK - A Tool Set for Automatic Performance Analysis of Parallel Applications,"** *Proc. of the European Conference on Parallel Computing (EuroPar)*, Springer-Verlag, Klagenfurt, Austria, LNCS 2790, pp. 1301-1304, August 26-29, 2003.

V. Getov, M. Gerndt, A. Hoisie, A. Malony, B.P. Miller, eds., **Performance Analysis and Grid Computing**, Kluwer Academic Publishers, Boston, Massachusetts, 2004.

T. Ludwig and B.P. Miller, eds., **On-line Monitoring Systems and Computer Tool Interoperability**, Nova Science Publishers, New York, 2003.

K.L. Karavanic and B.P. Miller, “A Framework for Multi-Execution Support Performance Analysis” in **On-line Monitoring Systems and Computer Tool Interoperability**, Nova Science Publishers, T. Ludwig and B.P. Miller, eds., New York, 2003.

P.C. Roth, D.C. Arnold, and B.P. Miller, **“Benchmarking MRNet: Measuring the Performance and Scalability of Tool Infrastructure”**, *High Performance Grid Computing Workshop* of the *International Parallel and Distributed Processing Symposium*, Santa Fe, NM, April 2004.

P.C. Roth and B.P. Miller, **“The Distributed Performance Consultant and the Sub-Graph Folding Algorithm: On-line Automated Performance Diagnosis on Thousands of Processes”**, *submitted for publication*, April 2005.

E.D. Collins and B.P. Miller, **“A Loop-aware Search Strategy for Automated Performance Analysis”**, *submitted for publication*, January 2005.

A.R. Bernat and B.P. Miller, **“Incremental Path Profiling”**, s*ubmitted for publication*, November 2003.

Bryan R. Buck, Jeffrey K. Hollingsworth, **A New Hardware Monitor Design to Measure Data Structure-Specific Cache Eviction Information**, *To Appear International Journal of High Performance Computing Applications*, 2005.

Chadd C. Williams, Jeffrey K. Hollingsworth, **Automatic Mining of Source Code Repositories To Improve Bug Finding Techniques,** *To Appear IEEE Transactions on Software Engineering*, 2005.

Bryan R. Buck, Jeffrey K. Hollingsworth, **Data Centric Cache Measurement on the Intel Itanium 2 Processor,** *Proceedings of SuperComputing 2004*, Nov. 2004.

Mustafa M. Tikir, Jeffrey K. Hollingsworth, **Using Hardware Counters to Automatically Improve Memory Performance,** *Proceedings of SuperComputing 2004*, Nov. 2004.

I-Hsin Chung, Jeffrey K. Hollingsworth, **Using Information from Prior Runs to Improve Automated Tuning Systems,** *Proceedings of SuperComputing 2004*, Nov. 2004.

Chadd C. Williams, Jeffrey K. Hollingsworth, **Bug Driven Bug Finders,** *International Workshop on Mining Software Repositories (MSR)*, May 2004.

Chadd C. Williams, Jeffrey K. Hollingsworth, **Interactive Binary Instrumentation,** *Second International Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, May 2004.

Bryan R. Buck, **Data Centric Cache Measurment Using Hardware And Software Instrumentation,** *Ph.D. Dissertation, University of Maryland*, June 2004.

Mustafa M. Tikir, Jeffrey K. Hollingsworth, **Efficient Instrumentation for Code Coverage Testing,** *International Symposium on Software Testing and Analysis (ISSTA) 2002*, July 2002.