U.S. Department of Energy, Office of Science, Advanced Scientific Research
Mathematical, Information, and Computational Sciences

Performance Technology for Tera-Class Parallel Computers: Evolution of the TAU Performance System

Final Report
DOE Agreement: DE FG03-01ER25501
(Project completion date: December 28, 2004)

Allen D. Malony

Computational Science Institute
Department of Computer and Information Science
University of Oregon

Introduction

In this project, we proposed to create new technology for performance observation and analysis of large-scale tera-class parallel computer systems and applications in this project. We set out to accomplish four goals. These were to provide in the TAU performance system:

1) greater dynamics and flexibility in performance measurements,
2) increased access to performance data during execution,
3) improved methods for performance mapping in multi-layered and mixed model software, and
4) more comprehensive application/system performance data integration.

These four areas defined the scope of our research efforts in the past three years. Our goals were to design technology solutions that would meet the performance observation and analysis requirements for tera-scale computation, and to demonstrate these solutions working in a robust, performance system made available in real TeraOPS DOE computing environments. We also promised to work in close association with Los Alamos National Laboratory and National Energy Research Scientific Computing Center to target both scalable system platforms and applications where the evolving TAU performance system could be proven and applied.

Significant progress was made to advance the capabilities of the TAU performance system covering the four work areas outlined in the contract proposal. The following sections describe in detail the results that were accomplished in the four work areas above and how these fit in with the work proposed in the renewal application.

Dynamic Performance Measurement Control

The TAU performance measurement system was extended to allow execution-time control and selection of the events to measure, the type of measurements to perform, and the conditions when to apply them. In the first phase of the project, we enhanced TAU's ability to define performance event groups and to control performance measurement at the group level. TAU's measurement system was updated to allow multiple metrics to be observed for each performance event, where the metrics values may come from hardware or software counters or timers. Important support for selective performance event measurement was also added. These results are described below.

Dynamic Grouping

TAU now provides a mechanism to partition the routines and statement-level timers into *profile groups*. A routine or a timer can belong to one or more groups. In earlier versions, TAU provided a command line option to enable or disable a group prior to executing the program. This restricted changes to the instrumentation at runtime. Also, a group was defined as a static entity and a user could only choose between a set of predefined groups.

With TAU's new performance event grouping mechanism, users can define arbitrary groups during instrumentation. The group is specified by a character string and applies to the entities to be instrumented at the file scope. The profile group is then registered at runtime and a dynamic association is created. This scheme was implemented in TAU's automatic instrumentation tools and it is backwards compatible with the statically defined profile groups. TAU's new profile browser, ParaProf, was also extended to support group based investigation of performance data. This allows a user to selectively mask a group from graphical displays.

Selective Instrumentation

TAU's automatic source instrumentation tool was updated to read an instrumentation specification file indicating what entities are to be instrumented. An "include" list of routines restricts instrumentation to only those specified entities, while an "exclude" list instruments all routines except those indicated. The user can also specify files for inclusion or exclusion. Support for wildcard characters that match one or more characters allows the user to specify lists with greater flexibility. Thus, selective instrumentation allows users to completely eliminate certain routines from consideration. It is typically used for controlling instrumentation in codes where light-weight routines are frequently executed. We developed the *tau_reduce* tool to automate the process of creating the selective instrumentation file using rules for disabling instrumentation. These rules are applied to historical performance data. Preliminary work was done on creating the TAU instrumentation specification language (TauIL) to interoperate with selective instrumentation. We intend to continue work on the TauIL. The instrumentation specification applies to TAU's automatic source instrumentation for C, C++, and F90, as well as to runtime instrumentation (e.g., using the Dyninst API). We demonstrated the use of selective instrumentation capabilities in Uintah, VTF and EVH1 (PERC) projects [1].

Multiple Counters

In previous versions of TAU, only one quantity at a time could be measured in an experiment (e.g., a timer or a hardware counter). This required more than one experiment to obtain a performance characterization of an application when multiple performance metrics were required. Unfortunately, both profiling and tracing had the restriction. TAU can now be configured to measure multiple quantities for a performance event in a single execution. Counters from hardware performance monitors (e.g., PAPI and PCL) as well as different execution metric sources (e.g., hardware cycle counter, wall clock timer, message sizes, low-level events from MAGNET/MUSE and Unix process time) can be simultaneous measured. In addition to these, an API allows for the addition of special purpose counter functions, including ones defined by the user. TAU can be configured to use as many counters as required, with no extra overhead when only one value is measured. Up to twenty-five counters can now be selected via environment variables for a performance measurement.

Perturbation Compensation

Significant advances were made to improve the accuracy of TAU's measurements. We implemented a scheme for evaluating the runtime overhead of TAU's measurement and eliminating this overhead. TAU supports a new configuration option *–COMPENSATE*, which installs a runtime overhead analysis module that calibrates the TAU measurement system to assess the cost of performing measurements at runtime and subtract this cost at runtime from the profile measurements. To do this, we needed to extend the data structures in TAU to keep track of descendent routines that are instrumented, for each routine. We obtained encouraging results from sequential overhead compensation and are now looking at portable techniques for parallel overhead compensation for profiling as described in [3].

Application-Level Performance Access

The project proposed to develop a programming interface in the TAU performance system to allow application threads of execution to directly access performance data at runtime. We developed three forms of application-level access support: incremental profile dumping, runtime profile access API, performance data management framework (PerfDMF), and online performance monitoring.

Incremental Profile Dumping

In previous versions of TAU, parallel profiles were only accessible for analysis at the end of program execution, after they were written to profile files. New functionality was implemented to allow the applications developer to trigger writing of the current parallel profile to an output file at any point during execution. This "incremental dumping" of profile data is available through an API that lets the user specify the list of routines (the default is all routines) for writing. Each time the dump routine is called, either a new dump file is generated or the profile dump is appended to earlier dump files. In both cases, a timestamp is recorded with the profile dump. The advantage of incremental profile samples is that they convey performance statistics for different time periods of a program's execution. Also, because the incremental profile data is available in the file system as the program is executing, it can be read by an external tool or interactively by the user. This is valuable in the case of large, long-running programs.

Runtime Profile Access

While the application can use TAU's incremental profile dumping to generate profile samples for later analysis, all file I/O is done by the profile library and the profile data itself is not accessible by the application. The ability for a program to query performance data directly at runtime will be important for application-level adaptive performance control. We developed a profile access API that allows the running application to query its profile information at runtime. Presently, the calling thread of control can only see its profile data, but this will change in the future. The API allows the thread to select all or part of its performance profile, down to individual function resolution.

TAU's performance access API was used for exploring computational quality of service issues [4] in component software at ANL and for developing performance models of CFRFS combustion code at SNL [8].

Performance Data Management Framework (PerfDMF)

We built the TAU performance data management framework (*PerfDMF*) [1] to support building cross-platform portable analysis tools for conducting multi-experiment performance studies. We developed a Java tool API that provides a profile data abstraction layer for use in developing analysis tools without requiring the tools to issue SQL commands. It is used by other projects such as *Jreduce*, to weed out fine-grained light-weight routines that execute frequently and perturb the application. Using this layer, TAU's profile browser, *ParaProf*, can connect to the PerfDMF and access performance data from different trials and experiments. We developed a PerfDMF browser tool which uses this interface to show the performance data for different trials as horizontal Gantt charts. Significant improvements were made during the last year to enhance the robustness of PerfDMF. Freely available databases such as PostgreSQL and MySQL as well as a commercially available database (DB2 from IBM) are now supported by TAU.

Online Performance Monitoring

TAU now supports online trace merging and conversion for online trace visualization. Our work last year with the Vampir Next Generation (*VNG*) [5] addresses scalability concerns with tracing. Using a cluster for distributed trace analysis of parallel trace data by separating the client visualizer from the parallel analysis server performs well for traces of large sizes. VNG accepts TAU's native binary trace format as it provides the flexibility of online-trace analysis that other static formats such as STF and VTF3 do not. We built the TAU trace input library [6] that parses the binary merged or unmerged traces (and their respective event definition files) and provides this information to an analysis tool using a trace analysis API. This API employs a callback mechanism where the tool registers callback handlers for different events. The library parses the trace and event files and notifies the tool of

events that it is interested in, by invoking appropriate handlers with event specific parameters as described in [6]. We intend to continue working on effective ways to visualize trace data.

Multi-Level Performance Instrumentation and Mapping

TAU offers the user a flexible instrumentation environment where different strategies can be used to insert instrumentation code in the program. These strategies range from manual and automatic instrumentation of source code, to instrumented libraries, to dynamic instrumentation of object and executable code, to virtual instrumentation in component software. Improvements were made in all of these aspects. In addition, we provided support to help control measurement overhead. These are discussed in the following sections.

Program Database Toolkit (PDT) and Source Instrumentation

PDT is the foundation for TAU's source instrumentation capabilities. It provides support for source analysis of C, C++, and Fortran 77/90/95 programs. In the past year, we added support for Fortran 95 instrumentation with the introduction of the new Flint parser from Cleanscape, Inc. We currently generate PDB files with enough information for automatically instrumenting Fortran source code for TAU instrumentation. Later this year, we plan to extend the parser to emit statement-level information as well. This capability greatly enhanced TAU's ability to instrument Fortran applications and this was demonstrated with Fortran applications such as SAGE from SAIC/LANL (joint work with Michael Gittings, SAIC), MCNP from LANL (joint work with Susan Post, LANL), and NAS Parallel Benchmarks, ESMF and OverFlow-D from NASA. We have released a new version of PDT (v3.2) to support this feature. We also added support for statement-level records for C99 and C++ languages in PDT. PDT was also ported to the IBM Power4 Linux (BlueGene/L) platform and we're working with LLNL on evaluating TAU and PDT on the BlueGene/L platform. We plan to extend the Fortran 95 parser to emit statement level information as records in PDB format as described in this proposal. We worked with Matt Sottile and Craig Rasmussen at LANL to apply PDT to CHASM and SILOON language interoperability projects to bridge the gap between Fortran [14] and C/C++ languages as it relates to scientific computing.

Dynamic Instrumentation

We worked with DyninstAPI developers at U. Maryland to integrate support for binary rewriting in TAU. *tau_run*, a mutator program can now instrument an application using DyninstAPI prior to execution or rewrite the in-memory instrumented image to disk to insert hooks to the TAU API at routine transition events. As more platforms and compilers are supported by DyninstAPI developers, this feature will prove to be quite useful given the difficulties of using dynamic instrumentation as a viable instrumentation option with batch queuing systems.

We also added support for *DynaProf* from U. Tennesee to support a TAU probe [1]. This provides an alternative API for instrumentation, and it gives the user the flexibility to choose DynaProf or TAU's tau_run for instrumentation, PAPI or TAU probes (which include PAPI's usage with TAU) for measurement, and TAU's *ParaProf* browser [2] for examining profile data. All measurement options such as callpath profiling, tracing are supported in this mode of operation. Further enhancements to TAU for supporting dynamic instrumentation in batch execution environments will be made.

Tracing Library Enhancements

TAU now supports the EPILOG trace generation library from the Research Centre Juelich (FZJ). We believe the EPILOG trace format has the flexibility and extensibility to handle all HPC event models. It currently supports full MPI 2.0 and OpenMP event models, and is currently being extended to more general threading event models. In addition to the event model advantages, EPILOG can supplement TAU's tracing library by supporting the generation of trace events with multiple count values and to gain access to more sophisticated trace analysis tools, such as FZJ's EXPERT. The main limitation of EPILOG is that it cannot be used for online performance monitoring in VNG for which a dynamic trace format such as TAU is superior to static event formats such as VTF3, EPILOG and Intel/Pallas STF. We plan to continue extensions to our trace format.

MPI Level Instrumentation

We enhanced TAU's MPI wrapper interposition library layer to include support for tracking message sizes for asynchronous communication operations in a portable manner. By keeping track of asynchronous requests and matching the requests at the completion of the synchronization operations in MPI Wait or Test routines, we now accurately report the message sizes involved in synchronization operations. Besides send and receive events, we also track sizes of messages in broadcast, reduce, scan, and other related MPI collective operations. TAU's user-defined events, developed earlier in the project are used for this purpose. Further extensions to the MPI interposition library layer will be made.

Mapping Performance Data

We introduced callpath profiling capabilities in TAU. The user can now specify a callpath depth as a runtime parameter. TAU uses a default callpath depth of 2. We evaluated the cost of generating callpaths and optimized our implementation of runtime lookups by using a specialized data structure to hold an array of pointers and callpath depth instead of using a string of unique names. This reduced the runtime overhead of callpath profiling by an order of magnitude. Coupled with advances in selective instrumentation and online-performance perturbation compensation, we present a case for callpath profiling as a viable measurement option [3]. Further improvements were made in reducing the size of profile files generated by TAU. This work was done in conjunction with Brian Miller at LLNL.

ParaProf Profile Browser

Several key improvements were made to the *ParaProf* profile browser. These include better support for grouping performance data, support for user defined events (in conjunction with MPI layer improvements), support for saving image data. A new *Gprof* style display was added to *ParaProf* to extract parent and child routine information in the callpath; to present the aggregate timings in a hierarchical manner for each routine for its immediate parents and children as shown in Figure 1 for a simulation from SAGE on 512 processors of QSC. It shows that of the three immediate parents of MPI_Waitall, TOKEN_MODULE::TOKEN_GS_R8 is the most significant. This display allows the user to navigate through the list of routines by clicking on their names. Some of these improvements were made based on feedback from the usage of TAU in SAMRAI and KULL groups at LLNL.

Trace to Profile Converter

We built a tool to convert VTF traces to TAU profiles. This converter allows a tool to specify a region of the trace to convert to a profile to provide a bridge between tracing and profiling. A trace visualization tool, for instance, can invoke this converter to generate aggregate performance data for a portion of the trace under consideration. ParaProf, can then visualize the profile data and do the analysis in conjunction with the trace visualization tool. Further extensions to extract profiling data from traces are described in this proposal.

System- and Hardware-Level Performance Integration

To associate system and hardware performance data with program-level events, we integrated mechanisms for accessing this data with TAU's portable performance measurement infrastructure. TAU can gather performance metrics from several sources such as the on-chip timers, virtual timers, the operating system interfaces, PAPI [7] for hardware performance counters, etc. We extended TAU to interface with other packages for extracting performance data. Future extensions to system and hardware level performance monitoring are described in this proposal.

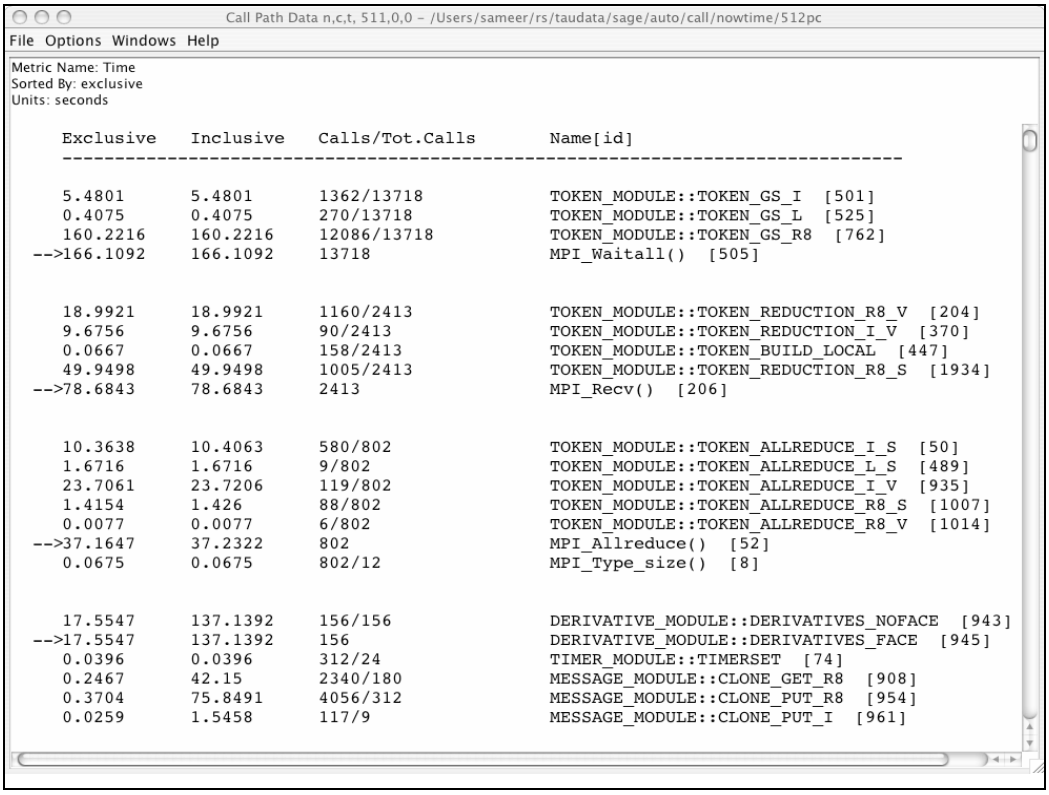

Figure 1 Callpath Display in ParaProf

Insights into the Kernel: MAGNET/MUSE

The kernel has a wealth of information that is not readily accessible to application level programs. MAGNET/MUSE from the Radiant group at LANL provides a mechanism for instrumenting the Linux kernel with a Kernel patch. It triggers events that are logged in a circular ring buffer accessible to the user –level MUSE process. TAU can interface with MUSE using AF_UNIX sockets and extract this low-level information and associate it with higher-level application events. We developed an interface with MAGNET/MUSE for both atomic events that occur periodically (via a timer interrupt) and start/stop timer events associated with routines and basic blocks. Information such as bandwidth rates associated with a process, context switches can now be seen in TAU.

Memory Profiling

We extended the TAU API to provide a heap memory utilization event. When the user enables tracking of this event, TAU generates a periodic interrupt. The user can control the interrupt interval and enabling and disabling of memory tracking at runtime. This work was done in conjunction with Andy Wissink from the SAMRAI project at LLNL who uses this functionality to track the progress of the adaptive mesh refinement. TAU's atomic user-defined events are used internally to implement this feature. Instead of sampling, TAU also allows memory inspection at specific locations in the source code by manual instrumentation. We can also use both interrupt-based and instrumentation-based schemes simultaneously. This work was done with Brian Miller and it is currently being applied to the MIRANDA project at LLNL.

To evaluate where memory allocation and de-allocation occurs in the source code, we implemented the TAU Memory interposition library that provides a wrapper for the C *malloc* and free system calls. By adding an option to the command-line, TAU's *malloc/free* library interposes between the application and the system call and keeps track of the location, the size of memory chunk associated with the call and the memory address referenced therein. TAU then presents summary statistics on the sizes of memory allocated/de-allocated along with call-site information in the form of source line number and file name. The library is portable across the different C

compilers and platforms and is independent of the underlying runtime system. This helps us track memory leaks in a program. Future extensions to this library will be made.

Conclusion

Portability, flexibility of instrumentation, and ease of use continue to guide TAU team's design decisions. Last year, we ported TAU and PDT to the IBM Power4 Linux (compatible with BlueGene/L) platform as well as the Mac OS X platform with IBM Visual Age C++/Fortran (xlC, xlf90) compiler suites. We also worked with Cray Inc. to port TAU and PDT to the Cray X1 as well as the Cray RedStorm Opteron Linux platform.

We worked closely with application groups and system administrators at LLNL, SNL, ANL, LANL to apply TAU effectively to their application codes and to periodically update their installations. TAU was qualified as a scalable performance tool on LANL's QSC machine and passed their validation testing. It is also supported on all platforms at NCSA and numerous other supercomputing centers around the world. We participated in the annual ACTS workshop at NERSC every year. We conducted tutorials and hands-on sessions at the ACTS workshop and demonstrated TAU at DOE sponsored booths at the annual SC conference every year. We intend to continue such user outreach activities as it provides us valuable user feedback.

Unexpended Project Funds

The current project received a no-cost extension through December 28, 2004. By this date, we did not have any unexpended project funds.

References

1. S. Shende, A. D. Malony, and R. Bell, "The TAU Parallel Performance System," (to appear) *International Journal of High Performance Computing Applications*, ACTS Collection Special Issue, 2004.
2. R. Bell, A. Malony, and S. Shende, "A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis," *International Conference on Parallel and Distributed Computing (EuroPar 2003)*, LNCS 2790, Springer, pp. 17–26, 2003.
3. A. Malony, and S. Shende, "Overhead Compensation in Performance Profiling," *International Conference on Parallel and Distributed Computing (EuroPar 2004)*, LNCS, Springer, to appear, 2004.
4. B. Norris, J. Ray, R. Armstrong, L. C. McInnes, D. Bernholdt, W. Elwasif, A. Malony, and S. Shende, "Computational Quality of Service for Scientific Components," *International Symposium on Component-based Software Engineering (CBSE7)*, LNCS 3054, Springer, pp. 264–271, 2004. Also available as Argonne National Laboratory preprint ANL/MCS-P1131-0204.
5. H. Brunst, W. Nagel, and A. Malony, "A Distributed Performance Analysis Architecture for Clusters," *IEEE International Conference on Cluster Computing (Cluster 2003)*, IEEE Computer Society, pp. 73–83, 2003.
6. H. Brunst, A. Malony, S. Shende, and R. Bell, "Online Remote Trace Analysis of Parallel Applications on High-Performance Clusters," *ISHPC'0*, LNCS 2858, Springer, pp. 440–449, 2003.
7. J. Dongarra, A. Malony, S. Moore, P. Mucci, S. Shende, "Performance Instrumentation and Measurement for Terascale Systems," *ICCS 2003*, LNCS 2660, Springer, pp. 53–62, 2003.
8. J. Ray, N. Trebon, S. Shende, R. Armstrong, and A. Malony, "Performance Measurement and Modeling of Component Applications in a High Performance Computing Environment: A Case Study," I*PDPS'04*, 2004. Also appears as Technical Report SAND 2003–8631, Sandia National Laboratories, Livermore, CA, Nov. 2003.
9. S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors," *International Journal of High Performance Computing Applications*, **14**(3), pp. 189–204, Fall 2000.
10. B. Buck and J. Hollingsworth, "An API for Runtime Code Patching," *Journal of High Performance Computing Applications*, **14**(4), pp. 317–329, Winter 2000.
11. L. Carrington, N. Wolter, and A. Snavely, "A Framework for Application Performance Prediction to Enable Scalability Understanding," *Scaling to New Heights Workshop*, Pittsburgh, PA, 2002.
12. R. Chandra, et al., *Parallel Programming in OpenMP*, Morgan-Kaufman, 2000.
13. R. Hornung and S. Kohn, "Managing Application Complexity in the SAMRAI Object-Oriented Framework," *Concurrency and Computation: Practice and Experience*, special issue on Software Architectures for Scientific Applications, 2001.
14. C. Koelbel, D. Loveman, R. Schreiber, G. Steele Jr., and M. Zosel, *The High Performance Fortran Handbook*, MIT Press, Cambridge, MA, 1994.
15. K. Lindlan, J. Cuny, A. Malony, S. Shende, B. Mohr, R. Rivenburgh, and C. Rasmussen, "A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates," *SC2000*, 2000.
16. A. Malony and S. Shende, "Performance Technology for Complex Parallel and Distributed Systems," in *Distributed and Parallel Systems From Instruction Parallelism to Cluster Computing*, G. Kotsis and P. Kacsuk (Eds.), Kluwer, pp. 37–46, 2000.

17. A. Malony, "Tools for Parallel Computing: A Performance Evaluation Perspective," in *Handbook on Parallel and Distributed Processing*, J. Blazewicz, K. Ecker, B. Plateau, an D. Trystram (Eds.), 2000, Springer-Verlag, pp. 342–363.
18. S. Parker, D. Weinstein, and C. Johnson, "The SCIRun Computational Steering Software System," in *Modern Software Tools in Scientific Computing*, E. Arge, A. Bruaset, and H. Langtangen (Eds.), Birkhauser Press, pp. 1–44, 1997.
19. PERC, DOE SciDAC Performance Evaluation Research Center. See http://perc.nersc.gov.
20. S. Shende, et al., "Portable Profiling and Tracing for Parallel Scientific Applications using C++," *ACM SIGMETRICS Symposium on Parallel and Distributed Tools*, 1998. See http://www.cs.uoregon.edu/research/paracomp/tau.
21. A. Malony, "Performance Observability," Ph.D. Dissertation, University of Illinois at Urbana-Champaign, Technical Report UIUCDCS-R-90-1603, October 1990.
22. C. Smith, *Performance Engineering of Software Systems*, Addison-Wesley, Reading, MA, 1990.
23. A. Snavely, et al., "A Framework for Performance Modeling and Prediction," *SC2002*, 2002.
24. J. D. de St. Germain, J. McCorquodale, S. Parker, and C. Johnson, "Uintah: A Massively Parallel Problem Solving Environment," *High Performance Distributed Computing Conference*, pp. 33–41, 2000.
25. VTF, Virtual Test Shock Facility, Center for Simulation of Dynamic Response of Materials. See http://www.cacr.caltech.edu/ASAP.
26. R. Prodan and T. Fahringer, "ZENTURIO: A Grid Service-based Tool for Optimizing Parallel and Grid Applications," (to appear) *Jounal of Grid Computing*, Kluwer Academic Publishers. http://www.par.univie.ac.at/project/zenturio/zenturio.html
27. The Omega Project for Statistical Computing. See http://www.omegahat.org/.
28. The R Project for Statistical Computing. See http://www.r-project.org/.
29. Weka Machine Learning Project. See http://www.cs.waikato.ac.nz/~ml/index.html.
30. CVS. See https://www.cvshome.org/.
31. Ptplot 5.3, Ptolemy II project, University of California, Berkeley. See http://ptolemy.eecs.berkeley.edu/java/ptplot5.3/ptolemy/plot/doc/index.htm.
32. DynaProf. See http://icl.cs.utk.edu/~mucci/dynaprof/.
33. L. DeRose, T. Hoover Jr, J.K. Hollingsworth, "Dynamic Probe Class Library – An Infrastructure for Developing Instrumentation for Performance Tools," Proc. IPDPS'2001 Conference. See http://oss.software.ibm.com/developerworks/opensource/dpcl.
34. MPI- Message Passing Interface (MPI) Standard. See http://www-unix.mcs.anl.gov/mpi.
35. W. Nagel, A. Alfred, M. Weber, H-C. Hoppe, S. Karl, "VAMPIR: Visualization and Analysis of MPI Resources," *Supercomputer* 63, Volume XII, Number 1, pp. 69-80, Jan. 1996.
36. B. Mohr, A. Malony, S. Shende, F. Wolf, "Design and Prototype of a Performance Tool Interface for OpenMP," *The Journal of Supercomputing*, **23**, 105-128, Kluwer, 2002.
37. IBM Corporation, "XL Fortran – Product Overview," See http://www.ibm.com/software/ad/fortran/xlfortran.
38. Intel Corporation, "Intel® Trace Collector," See http://www.intel.com/software/products/cluster/vampirtrace/.
39. S. Seidl, "VTF3 – A Fast Vampir Trace File Low-Level Management Library," Technical Report ZHR-R-0304, Dresden University of Technology, Dresden, Germany, Nov. 2003.

40. F. Song, F. Wolf, “CUBE User Manual,” ICL Technical Report IC-UT-04-01, University of Tennessee, Feb. 2004.

41. Eclipse Tools Project, See http://www.eclipse.org.

42. GNU Emacs Lisp Reference Manual, See http://www.gnu.org/software/emacs/elisp-manual/.

43. The PperfDB Project, See http://www.cs.pdx.edu/~karavan/pperfdb.html.

44. GNU Operating System – Free Software Foundation (FSF). See http://www.gnu.org.

45. S. Hackstadt, A. Malony, B. Mohr, “Scalable Performance Visualization for Data-Parallel Programs,” Proc. Scalable High Performance Computing Conference,” pp. 342-349, 1994. See http://www.cs.uoregon.edu/research/paracomp/papers.

46. M. Heath, A. Malony, D. T. Rover, “The Visual Display of Parallel Performance Data,” *IEEE Computer*, 28(11), pp. 21-28, Nov. 1995.

47. M. Heath, A. Malony, D. T. Rover, “Parallel Performance Visualization: From Practice to Theory,” IEEE Parallel and Distributed Technology, 3(4), pp. 44-60, Winter 1995.

48. A. Malony, S. Shende, R. Bell, K. Li, L. Li, N. Trebon, “Advances in the TAU Performance System,” Chapter, *Performance Analysis and Grid Computing*, (Eds. V. Getov, M. Gerndt, A. Hoisie, A. Malony, B. Miller), Kluwer, Norwell, MA, pp. 129-144, 2003.