Application Specific Performance Technology for Productive Parallel Computing

Final Project Report (DOE Agreement: DE-FG02-05ER25680) (April 1, 2005 – September 30, 2008)

Allen D. Malony	Sameer Shende
Department of Computer	Performance Research Lab
and Information Science	University of Oregon
University of Oregon	Eugene, OR 97403
Eugene, Oregon 97403	

1 Introduction

Our accomplishments over the last three years of the DOE project “Application-Specific Performance Technology for Productive Parallel Computing” (DOE Agreement: DE-FG02-05ER25680) are described below. The project will have met all of its objectives by the time of its completion at the end of September, 2008. Two extensive yearly progress reports were produced in March 2006 and 2007 and were previously submitted to the DOE Office of Advanced Scientific Computing Research (OASCR). Following an overview of the objectives of the project, we summarize for each of the project areas the achievements in the first two years, and then describe in some more detail the project accomplishments this past year. At the end, we discuss the relationship of the proposed renewal application to the work done on the current project.

2 Project Objectives

The current project proposed to create performance technology that improves the observation and understanding of parallel performance with respect to application-specific concerns. The project targeted new developments in the TAU parallel performance system [11] and the Program Database Toolkit (PDT) [3] that we believed would lead to significant advances in performance measurement and analysis capabilities with application-specific enhancements. The specific areas where we proposed work were:

- **Performance instrumentation** to enhance the specification and management of performance events by users and allow instrumentation selection guided by performance analysis.
- **Source code analysis** to improve PDT’s cross-language analysis, analysis resolution, and support for performance data mapping.

- **Customizable performance measurement** to give developers more programmatic and runtime control over the type, scope, and content of performance measurements, including ability to track execution path, data and memory usage, and systems interactions.
- **Performance tracing** to include storage of multiple counters in trace events, translation of TAU trace format to Epilog [13] and slog2 [14], and generation of TAU profiles from TAU and VTF [6] traces.
- **Performance data management and data mining** to make PerfDMF [1] a significant core component in TAU for integration with performance analysis tools, and to develop new performance data mining and discovery capabilities.
- **Scalable performance visualization** to view multi-dimensional temporal and spatial performance information from large-scale experiments to gain meaningful insight.

Cross-cutting each of these areas, the project placed a high emphasis on improving the overall usability of TAU.

The project has been a tremendous success in all areas. Our thesis of improving performance understanding through application-specific support in the performance tools has been proved true. The productive benefits of our project accomplishments can be seen in the increased use of TAU in DOE HPC community.

3 Performance Instrumentation and Source Analysis

Application-specific performance measurement and analysis requires an ability to instrument those events of the most interest to developers. Our work in the first two years of the project advanced instrumentation capabilities in several ways:

- **Eclipse integration.** TAU instrumentation plugins were developed for the Eclipse [7] framework to work with the Parallel Tools Platform (PTP). TAU instrumentation can be specified through Eclipse PTP, viewed with respect to the source, and translated into instrumented code for C, C++, and Fortran 90 programs.
- **Python instrumentation.** TAU was extended to support automatic instrumentation for Python using the latest features of Python’s debugger, which automatically detects routine and method calls and invokes callbacks to subscribers of those events. The instrumentation also supports routines called by Python code. Python-based TAU instrumentation was demonstrated with SciPy [10], Numpy [8], and PyACTS, a collection of Python packages for numerical computation based on the DOE ACTS toolkit collection [9].
- **Wrapper instrumentation.** A PDT-based tool was developed to automatically generate wrappers to routines for purposes of instrumentation. This has been applied for capturing events in thread libraries, memory allocation, I/O, and so on.
- **Outer loop instrumentation.** Support was added to PDT for automatic outer loop level instrumentation in C, C++, and Fortran. The techniques even work correctly with *goto* statements. TAU’s selective instrumentation was also extended to specify outer loop events.
- **Throttling instrumentation.** To eliminate lightweight events with high measurement overhead, we developed an online overhead analysis and event throttling mechanism to turn off

these events at runtime. Another tool then parses the TAU performance data output to identify which events were throttled and generate a selective instrumentation file to exclude those events from being instrumented in the future.

- **Instrumentation using library preloading.** TAU now can enable instrumentation in libraries at load time using preloading mechanisms on Linux and IBM platforms. Using this feature, TAU can avoid source code modification and recompilation. This has proven to be a significant enhancement to TAU’s usability.

The work leveraged advances in PDT capabilities for source code analysis. PDT’s Fortran 95 front-end based on Cleanscape parser was improved for error handling and finer-grained instrumentation. We integrated the GNU gfortran parser in the PDT distribution for the main HPC platforms. This is a significant advance for the project since it is the second open source parser for Fortran used by PDT. (The first was the Open64.) Gfortran is important because of the GNU community involvement and its broad distribution with GCC. Another major milestone was achieved with the upgrade of the Edison Design Group (EDG) front-end parser for C and C++ from v2.45 to v3.8, EDG’s latest release. This significantly improves the robustness of PDT to handle the latest C/C++ language specifications.

In the last year, we added support for memory leak instrumentation, updated Eclipse/PTP for accessing TAU options and build configurations, ported the gfortran parser to the Mips Linux architecture for use in the SiCortex platform, and enabled signal handlers to be used for toggling instrumentation. The TAU wrapper technology was improved for external libraries and new wrappers for POSIX I/O and MPI-I/O were added for tracking the volume and bandwidth of I/O. Finally, TAU’s instrumentation specification approach now allows static/dynamic phase/event instrumentation constructs to be included.

4 Performance Measurement

We significantly extended the application-specific support in the TAU performance measurement system. The most prominent enhancement in the first two years was the addition of *phase profiling* [4]. We implemented an API to annotate application specific phases. Phase profiling helps map performance profiles to phases of computations. Phases can be used to represent structural, logical, and execution time aspects of the program, and phases may be nested. Phases are identified in the source code using static or dynamic phase annotations. Static phases track performance (profile) of all routines called directly or indirectly in a phase, over all invocations of the phase. Dynamic phases, by contrast, track this data for each invocation of the phase. When phase profiling is turned off, the phase annotations in the source code are mapped to the corresponding standard event calls. Phase-based profiling was applied to several projects including the FLASH, S3D, MFIX, and Uintah.

We also added other profiling capabilities. In Year 1, callddepth profiling was added to TAU’s callpath profiling technology. For some performance evaluation studies, it is desirable to see how the performance is distributed across program parts from a top-down, hierarchical perspective. Thus, a parallel profile that showed how performance data was distributed at different levels of an unfolding event calltree could help application developers better understand the performance. TAU’s implementation of callddepth profiling does just that. It allows the user to configure TAU with the -DEPTHLIMIT option and specify in the environment variable how far down the event calltree to observe performance. It works with both flat as well as callpath profiles.

Context events were also developed in Year 1. TAU’s user defined atomic event names appear independent of the routine in which the events were triggered. Scientists at the ANL needed a

solution to track events and associate these events with the executing callpath. They needed this functionality without incurring the overhead of callpath profiling of interval events. We designed a context event API in TAU that allows users to trigger events with data. The data is then associated with an atomic event whose name embeds a portion of the currently executing callpath.

In Year 2, we added parameter-based measurement to create events dynamically based on program execution behavior and semantics. It is parameter-based in the sense that the parameters of a routine could be used to define a new event. This is useful in workload characterization to differentiate the performance of a routine with respect to how it is used. We have applied this to performance measurement of MPI routines based on message size. The programmer can use any parameter or combination of parameters for defining events. Parameter-based measurement is being used by the SciDAC Center for Technology for Advanced Scientific Component Software (TASCS) for implementing computational quality of service (CQoS) support.

Other performance measurement achievements in the first two years include tracking nested OpenMP events, MPI-2 support, capturing of runtime metadata, Cray XT3 measurement support, and Perflib integration.

This past year continued the strong development success. The work on profile snapshots [5] in Year 2 was completed. A “profile snapshot” is a view of the current parallel profile at a point during execution. The TAU measurement system has long had the ability to “dump” performance profiles to files during execution. A profile snapshot can be selective such that only a subset of the performance events being measured are queried, and the event subset may change from snapshot to snapshot. Profile snapshots are completely under the control of the application, in terms of where they are taken and when. This allows developers to better specialize performance observation to their needs.

Other achievement include enhancements for PAPI, support for persistent communication events in traces, ports to new platforms (Cray XT4 Compute Node Linux, IBM BG/P, SiCortex, Windows Cluster 2003 (including MPI support)), new compiler support (Intel 10.x Fortran/C/C++, NAGWare Fortran, and g95 Fortran),

5 Performance Tracing

Our project objectives for tracing were to update TAU with state-of-the-art tracing capabilities and to put TAU in position to transition to an integrated solution with Scalasca. In the first year, we developed support for recording user-defined atomic events in trace files. At each event entry and exit, an array of counter values can be collected and stored with the event. With the latest features of tools such as Vampir, these counter values can be see over time and views as a rate.

We also focused on a scalable trace merging tool to avoid problems with limits on the number of trace file descriptors that can be simultaneously open. Our solution is hierarchical and is based on timestamp sequencing. While merging traces, timestamps are compared from each tracefile to determine which record is written to the merged tracefile where records are sorted by time. TAU’s tracing infrastructure was re-engineered to allow traces to be generated on one platform and analyzed on another. Generating platform independent tracefiles and tools maintains compatibility with older traces.

For TAU traces to be analyzed, they have to be converted to formats recognizable by the trace analysis tools. In Year 1 we added trace converters for SLOG2 (*tau2slog2*), VTF3 (*tau2vtf*, EPILOG (*tau2elg*), and OTF (*tau2otf*). SLOG2 is used with the freely available Jumpshot trace visualizer (the SLOG2 SDK and Jumpshot are shipped with TAU) and EPILOG and OTF are supported in Scalasca. In addition, we developed *tau2profile* to extract profiles from TAU traces. These profiles

can then be viewed using TAU’s ParaProf profile browser.

In Year 2, we released the libraries for reading and writing OTF files were released as open source. TAU was updated to generate OTF files either through conversion from the TAU trace format, or the EPILOG tracing library with the OTF output option. The Vampir and Vampir Server trace analysis and visualization tools read OTF directly. We also developed a pure Java version of our TAU trace writer library.

The main improvement for parallel tracing in the past year has been the implementation of clock synchronization support both at runtime and through post-mortem trace analysis. This has proven to be of high value importance as part of our activities with several DOE SciDAC projects. In addition, we used our pure Java trace writer library to developed a pure Java implementation of *tau2slog2* tool.

6 Performance Data Management and Data Mining

Significant advances have been made in the technology TAU uses to store parallel profile data and conduct multi-experiment performance analysis. The first two years saw improvements in PerfDMF to increase its robustness. This included the adding of multiple backend database engines and improving its external interfaces for use by ParaProf and other tools. For instance, PerfDMF initially required an SQL database backend for cross-experiment data analysis. We supported Oracle, PostgreSQL, and MySQL. In year 2, we were able to configure PerfDMF to use a file system database called Derby. The use of Derby enables database capabilities to users who are unable to install and maintain a proper SQL server. In addition to Derby, we also added support for DB2 in Year 2. The ParaProf and PerfExplorer tools now fully support all database engines through PerfDMF.

The big achievement in Years 1 and 2 was the creation of PerfExplorer [2], a framework for parallel performance data mining and knowledge discovery. The framework architecture enabled the development and integration of data mining operations for multi-dimensional, large-scale parallel performance profiles. We were able to provide access to the R statistics system and Weka data mining package utilizing the framework support. In the second year of the project, several new features were added. Users could create normal probability plots of experiment trial data, adjust the dimension reduction value to control the number of events in a plot, browse atomic event data in a trial, and get detailed information about user-defined events.

PerfExplorer was initially developed as a client-server system, utilizing PerfDMF to access the parallel profiles and save its analysis results. A user interacted with the first version of PerfExplorer through a GUI, interactively choosing from a set of statistical analysis and visualization functions and applying them to selected multi-dimensional, multi-experiment performance data sets. While effective for standard comparative analysis and clustering, PerfExplorer’s software design made it difficult to extend and link together multiple data mining steps. We began in Year 2 to re-engineer PerfExplorer to use an internal scripting engine based on Jython, a native Java implementation of Python. The goal is to make it possible to program new data mining operations and to link them into data mining workflows.

This past year has seen an increase in the use of both PerfDMF and PerfExplorer. PerfDMF was already in use by several application groups, but started to be used by the PERC/PERI SciDAC projects. This motivated important PerfDMF extensions in the areas of metadata schema and storage, integration with the TAU portal, XML output generation, and additional profile importers. In addition, we added support in PerfDMF for phase and profile snapshots, OMPP and CUBE-3 profile files, and multiple databases. For PerfExplorer, we continued the re-engineering work and

began leveraging the Jython interpreter support. We opened up access to PerfExplorer for scripting. New comparative analysis and displays were implemented. The ability to create custom charts with XML metadata fields was also added. More interestingly, we are building in support for analysis rules and inference processing, so that PerfExplorer can be used for decision support.

7 Scalable Performance Visualization

Our work in scalable performance visualization was directed toward problems viewing multi-dimensional performance information from large-scale experiments. We targeted our work mainly to ParaProf, but also used data mining for dimensionality reduction. In the first two years, we developed scalable performance displays using two techniques: histogramming and 3D graphics. Scalable performance histograms bin performance data for multiple threads of execution on one metric dimension and then show the distribution of numbers of threads in each bin. The number of bins is adjustable and the metric range is uniformly divided.

Our 3D visualization work developed surface plots to show full performance profiles and four-degree scatter plots to focus in on four metrics for analyzing thread clustering relationships. We added a stereo visualization option to be used when a 3D stereo display is available. 3D stereo visualization of 16K cpu parallel profiles were demonstrated for the first time at SC 2006 on the large format display in the DOE ASCI booth. The 3D visualization capabilities are utilized by both ParaProf and PerfExplorer.

In the past year, we have focused on applying scalable performance visualization for large-scale parallel performance analysis. To get a brief sense of how this feature is useful, Figure 1 shows 3D views of the performance of S3D [12], a high-fidelity finite difference solver for compressible reacting flows, running on a hybrid Cray system consisting of XT3 and XT4 processing nodes. The performance data is of a S3D execution running on 6400 cores. ParaProf analyzed the full profile and generated the three-dimensional view shown in Figure 1, left display. Due to the XT3 and XT4 memory performance differences, imbalances resulted in the S3D computation. These are clearly apparent in the ParaProf display. The right display in the figure is a ParaProf scatterplot showing clustering relationships for three significant events, color-code by processing core type.

Figure 1: ParaProf displays of S3D performance on hybrid Cray XT3/XT4 system.

8 Conclusions

Our highly successful work on the present project has clear, direct relevance to future work in the TAU project. First, without the advances we achieved in the last three years, the TAU performance system would not be at the level of maturity and sophistication it is at presently. This means that the TAU technology to refactor and integrate with Scalasca is coming in at a significantly higher entry point. There are major new additions to the TAU performance system realized as a result of our work on the present project that makes the new project practically possible.

Second, although it was not part of our direct objectives, there have been interactions with the Kojak and Scalasca teams throughout the past three years that put us in a solid initial position for joint work in the future. This is evidence in many aspects of the project, from importing of CUBE files in PerfDMF, to TAU trace conversion to EPILOG and OTF, to even patterning ParaProf's hierarchical display after the CUBE browser. The communication between the two groups has been constant throughout the project and has allowed each group to stay abreast of developments in the other project.

Lastly, the structure of the new project is consistent with the present one in focusing on the core areas of instrumentation, measurement, and analysis. The results we have achieved will, in a real sense, be the starting point for the refactoring and re-engineering activities in the TAU and Scalasca project towards a unified measurement substrate.

References

- [1] K. Huck, A. Malony, R. Bell, and A. Morris. Design and Implementation of a Parallel Performance Data Management Framework. In *Proceedings of the International Conference on Parallel Computing, 2005 (ICPP2005)*, pages 473–482, 2005.
- [2] K. A. Huck and A. D. Malony. Perfexplorer: A Performance Data Mining Framework for Large-Scale Parallel Computing. In *Conference on High Performance Networking and Computing (SC'05)*, Washington, DC, USA, 2005. IEEE Computer Society.
- [3] K. A. Lindlan, J. Cuny, A. D. Malony, S. Shende, B. Mohr, R. Rivenburgh, and C. Rasmussen. A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates. In *Proceedings of SC2000: High Performance Networking and Computing Conference*, 2000.
- [4] A. Malony, S. Shende, and A. Morris. Phase-based Parallel Performance Profiling. In *PARCO'05: Conference on Parallel Computing*, September 2005.
- [5] A. Morris, W. Spear, A. Malony, and S. Shende. Observing Performance Dynamics using Parallel Profile Snapshots. In *Europar'08: European Conference on Parallel Processing*, 2008.
- [6] W. Nagel, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 63, XII(1):69–80, 1996.
- [7] The Eclipse Project. Eclipse. <http://www.eclipse.org/>.
- [8] The NumPy Project. Numpy. <http://numpy.scipy.org/>.
- [9] The PyACTS Project. Pyacts. <http://www.pyacts.org/>.
- [10] The SciPy Project. Scipy. <http://www.scipy.org/>.

- [11] S. Shende and A. D. Malony. The TAU parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–331, 2006. SAGE Publications.
- [12] R. Subramanya and R. Reddy. Sandia DNS code for 3D compressible flows - Final Report. Technical Report PSC-Sandia-FR-3.0, Pittsburgh Supercomputing Center, PA, 2000.
- [13] F. Wolf and B. Mohr. EPILOG Binary Trace-Data Format. Technical Report FZJ-ZAM-IB-2004-06, Forschungszentrum Jülich, May 2004.
- [14] C. Eric Wu, Anthony Bolmarcich, Marc Snir, David Wootton, Farid Parpia, Anthony Chan, Ewing Lusk, and William Gropp. From trace generation to visualization: A performance framework for distributed parallel systems. In *Proc. of SC2000: High Performance Networking and Computing*, November 2000.