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

Performance Engineering Technology for Scientific Component Software

Final Report: 8/05/03 – 2/01/07

DOE Agreement: **DE-FG02-03ER25561**

Allen D. Malony

Department of Computer and Information Science
University of Oregon

1. Introduction

Large-scale, complex scientific applications are beginning to benefit from the use of component software design methodology and technology for software development. Integral to the success of component-based applications is the ability to achieve high-performing code solutions through the use of performance engineering tools for both intra-component and inter-component analysis and optimization. Our work on this project aimed to develop ***performance engineering technology for scientific component software*** in association with the DOE CCTTSS SciDAC project (active during the contract period) and the broader Common Component Architecture (CCA) community. Our specific implementation objectives were to extend the *TAU performance system* and *Program Database Toolkit (PDT)* to support performance instrumentation, measurement, and analysis of CCA components and frameworks, and to develop performance measurement and monitoring infrastructure that could be integrated in CCA applications. These objectives have been met in the completion of all project milestones and in the transfer of the technology into the continuing CCA activities as part of the DOE TASCS SciDAC2 effort. In addition to these achievements, over the past three years, we have been an active member of the CCA Forum, attending all meetings and serving in several working groups, such as the CCA Toolkit working group, the CQoS working group, and the Tutorial working group. We have contributed significantly to CCA tutorials since SC'04, hosted two CCA meetings, participated in the annual ACTS workshops, and were co-authors on the recent CCA journal paper [24].

There are four main areas where our project has delivered results: component performance instrumentation and measurement, component performance modeling and optimization, performance database and data mining, and online performance monitoring. This final report outlines the achievements in these areas for the entire project period. The submitted progress reports for the first two years describe those year's achievements in detail. We discuss progress in the last project period in this document. Deployment of our work in CCA components, frameworks, and applications is an important metric of success. We also summarize the project's accomplishments in this regard at the end of the report. A list of project publications is also given.

2. Component Performance Observation (Instrumentation and Measurement)

The principal objective in our work was to develop support for observing the performance of scientific component applications. It was necessary to do so in a manner consistent with the component software

engineering development methodology and tools. We created an interface for component software for performance instrumentation and measurement and a performance observation component that implements this interface in a manner compatible with CCA software methods and frameworks. The CCA *performance component* provides interfaces for event creation, measurement options, data query, and runtime control. Underneath the performance component interface is an implementation that provides access to the full capabilities of the TAU performance system, including parallel profiling and tracing. Our purpose for this approach was to abstract the performance interface that developers of CCA applications would use for performance observation from the underlying measurement facility, allowing alternative measurement infrastructure to be accessed. The performance component can be used with different CCA frameworks (SIDL and C++) and is included as part of the CCA Toolkit distribution.

The CCA performance component can be used to instrument components used in an application. In addition, we developed an approach to instrument component interfaces so that the performance along an edge of the component interconnection graph (between a caller and callee connecting Uses and Provides ports) can be tracked. This is done through the use of *proxy components,* which are interposed between a caller and callee, and capture performance data for each port of the components. Proxy components use the performance component for measurement. With our program database toolkit (PDT), we implemented support for automatic proxy instrumentation by automatically creating proxy components from component source analysis. In addition, we created a Python SIDL parser to read a component's SIDL specification and automatically generate the C++ proxy server code. The proxy generators for instrumentation of classic C++ and SIDL components are bundled in the TAU distribution for CCA.

Our accomplishments are summarized below, followed by more in-depth discussion of recent progress in performance instrumentation and measurement.

- Parallel performance observation of component applications fully integrated with CCA distribution
- Definition of abstract performance component for CCA instrumentation and measurement
- Proxy component approach for component interface instrumentation and measurement
- Automatic proxy component generation
- Performance and proxy component technology available for classic C++ and SIDL components
- Full access to TAU performance measurement capabilities

2.1 Formulation of component performance ports

The generic ports of the performance component allow for any performance tool to be used by a CCA component framework. The "measurement port" defines the *Timer*, *Phase*, *Event*, *ContextEvent*, *Query*, and *MemoryTracking* interfaces [24]. Depending on the performance infrastructure available, some or all of the interfaces may be active.

The Timer interface is used to bracket a region of code with start and stop methods and gather performance data. The Timer definition was recently extended to allow for parameterization using TAU's `-PROFILEPARAM` option. Here, a timer can be associated with a key and a value, allowing performance data to be partitioned based on application parameters. For instance, the time spent in MPI routines can now be partitioned based on the message size, helping to characterize the effects of the communication subsystem for a component based.

The performance component interface supports both profiling and tracing measurement options. Profiling records aggregate inclusive and exclusive wall-clock time, process virtual time, hardware performance metrics such as data cache misses and floating-point instructions executed, as well as a combination of multiple performance metrics. The Phase interface provides access to TAU's phase measurement support [13]. Phases are similar to timers in their usage and methods. However, phases make it possible to correlate the performance of one part of the program to another. They borrow concepts of performance mapping found in our earlier work on callpath profiling. In phase measurements, all performance data measured between phase entry and exit is associated in its entirety with the phase.

The Event interface enables the use of TAU's atomic user-defined event to track application and system level events that are not associated with a start and stop event such as memory utilization and the extent of inter-process message communication. The ContextEvent interface, recently added, enables the use of TAU's powerful ability to associate user-defined atomic events within the context in which they are triggered, similar to callpath profiling. This way, the data supplied is distinguished based on the location where an event occurs and the sequence of actions (timer invocations) that preceded the event. The depth of the callstack embedded is configurable at runtime.

The Query interface provides access to runtime performance data collected by TAU. It provides access to both timer based events as well as atomic user defined events. The Control interface allows the user to control the instrumentation by enabling or disabling one or more groups of timers and provides runtime instrumentation control capabilities to components. For example, a user can enable or disable all MPI timers via their group identifier. The MemoryTracking interfaces provide access to TAU's memory utilization and headroom tracking support [14].

Though the performance port definitions are generic, the TAU-based performance component is the primary implementation for CCA. We continue to ensure that the performance component and all related tools are updated to the most recent CCA conventions. Recently, all the tools were updated to use the new C++ Babel binding, thus ensuring the continuation of performance tool availability for CCA application developers.

2.2 Construction of automated proxy component generators

If a CCA application developer wants to capture performance associated with component use, proxy components can be used. For each component that the user wants to analyze, a proxy component is created. The proxy component shares the same interface as the actual component. When the application is composed and executed, the proxy is placed directly "in front" of the actual component. Since the proxy implements the same interface as the component, the proxy intercepts all of the method invocations for the component. In other words, the proxy uses and provides the same types of ports that the actual component provides. In this manner, the proxy is able to snoop the method invocation on the Provides Port, and then forward the method invocation to the component on the Uses Port.

We define two types of proxies. M*easurement proxies* connect directly to the TAU performance component through a Measurement port. These proxies bracket each port invocation on the attached component with a start and stop call to the TAU performance component. In this way, any measurement capability that the TAU library supports can be captured by the proxy.

The other kind of proxy, the *monitor proxy*, connects to a *monitoring component*, like the mastermind component described later. Though it also brackets the actual component port call with start and stop events, it differs from the measurement proxies in that it also marshals many of the port parameters and their names through the Monitor port. We presently can marshal all primitive values, such as doubles and integers, as well as arrays.

2.3 High-level TAU measurement API

The CCA performance component provides an abstract measurement interface for CCA developers to use while hiding the measurement implementation. Component developers can manually instrument their code using the performance component interface. We also provide automatic instrumentation support in TAU. To abstract instrumentation further to make it less CCA dependent and allow for infrequent manual instrumentation, we introduced a high-level measurement API in the latest TAU release. Two routines are defined:

- `TAU_START("func_name")`
- `TAU_STOP("func_name")`

When invoked in a CCA component, the calls are properly linked by TAU to the performance component. The benefit to the user is the ability to instrument code modules in components with instrumentation that is reusable when the modules are used in other non-CCA programs. However, this instrumentation should only be used for coarse-grained instrumentation, as it is less efficient than TAU's automatic instrumentation for CCA.

3. Component Performance Modeling and Optimization

An important goal of the project was to provide initial support for computational quality of service (CQoS) in scientific component applications [18]. This required a robust performance instrumentation and measurement infrastructure integrated in the CCA programming environment. In addition, we wanted to demonstrate an ability to create performance models based on measurements in relation to the application context, component-specific parameters, and component interactions. The goal was accomplished in both online and offline modes.

The early project work focused on the creation of a *mastermind* component based on the proxy component technology [26, 23]. The mastermind component monitors component methods and tracks performance for each invocation. From this information, a performance model is generated to predict an individual component's performance in the application context. This work was done in conjunction with Jaideep Ray at Sandia National Laboratory on the CFRFS application. Enhancements of the mastermind component were made to model per-component usage and to enable tracking of performance associated with each edge of the component interconnection graph when multiple instances of components and their ports offer variations of the core functionality.

Significant results were delivered in this area of the project, including the following:

- Empirical modeling of parallel component performance parameterized by application context
- Development of mastermind component for component model measurement
- Support for component usage modeling based on caller and method parameters
- Application of performance modeling to CCA applications, mainly CFRFS
- Initial support for runtime performance model evaluation
- Mastermind component technology available for classic C++ and SIDL components

Recent accomplishments are reported below.

3.1 Online introspection

The mastermind component has been extended to provide an additional interface that permits runtime examination of the detailed performance and application specific method argument data generated. This online introspection will permit performance expectations, based on previously derived models, to be evaluated and tested at runtime. Together with the online monitoring support being developed, this will provide a foundation for CQoS during execution.

3.2 Component ensemble modeling and optimization

With a performance evaluation infrastructure in place for component-based applications and a working performance modeling system, we developed methods to investigate the problem of optimizing component assemblies. Implementation of techniques for automatic proxy creation of both mastermind and performance components for both classic C++ and SIDL interfaces made it possible to analyze an assembly of different types of components. The approach first generates lightweight performance proxies for all components to determine which ones contribute significantly to the overall runtime. For this subset of components, mastermind proxies are created and their performance models are generated using techniques developed within our group [2] and by other CCA researchers at LANL and SDSC.

To optimize a component assembly given different implementations for components, the problem of choosing the correct sets of components is an open research problem. Using all possible combinations for

each component would yield a combinatorial explosion of choices. To address this problem, we developed a tree pruning scheme. This scheme takes as its input the callgraph generated by the mastermind proxy component and a set of rules specified by the user. It then prunes the callgraph and removes all nodes deemed insignificant to the optimization. With the reduced callgraph, it is now possible to possible combinations in the reduced set to arrive at an optimal solution.

We built a modeling library that takes as its input routines that describe the performance model of a component. Currently, a performance model is embedded in the library. This allows the user to create arbitrarily complex performance models in the form of routines that return a performance metric given a set of parameters. The performance models could be described as algebraic expressions of input arguments or use a curve fitting approach to estimate the performance from historical performance data points. We created an *optimizer* component that uses the modeling library to try all possible permutations of component instances, based on a user-specified performance model for each component. Components are described in a hierarchical fashion using the concept of component families where different implementations of a given component perform equivalent functions and may be may be substituted. The optimizer component will select an optimal component assembly given pruning heuristics and measurement-based performance tuning.

The component modeling and optimization techniques above are implemented in the TAU distribution for use in CCA applications [25, 1].

4. Performance Database and Data Mining

Empirical performance evaluation of parallel component-based applications can generate significant amounts of performance data and analysis results from multiple experiments as performance is investigated and problems diagnosed. To better manage performance information, we developed the Performance Data Management Framework (*PerfDMF*) [7] shown in Figure 1. PerfDMF stores and manages parallel performance profiles in a relational database backend while providing a high-level query interface to facilitate the creation of performance analysis tools. Our research and development work on PerfDMF took place throughout the project, allowing for its evaluation in the context of CCA performance analysis needs. PerfDMF has been used extensively within the CCTTSS SciDAC project. especially for CQoS investigations [18]. Here, we are working closely with Boyana Norris and Lois Curfman McInnes at the Argonne National Laboratory. Our work with Sandia described above also uses PerfDMF to store multi-experiment performance data from which performance models are derived.

Our accomplishments in parallel profile database work are listed below:

- Robust relational database implementation for parallel performance profiles
- Support for multiple backend database systems, including in-memory Derby DB
- Java, C, and C++ query interface for performance analysis tools
- Unified parallel profile data model
- Multiple importers for parallel profile data produced by other tools
- ParaProf (TAU's parallel profile analysis tool) integration with PerfDMF
- Secure database access and support for distributed operation
- Flexible schema customized to component metadata and measurement requirements

During the last project phase, we have leveraged significantly the success of PerfDMF for parallel performance data mining. The following describes our PerfExplorer work in detail.

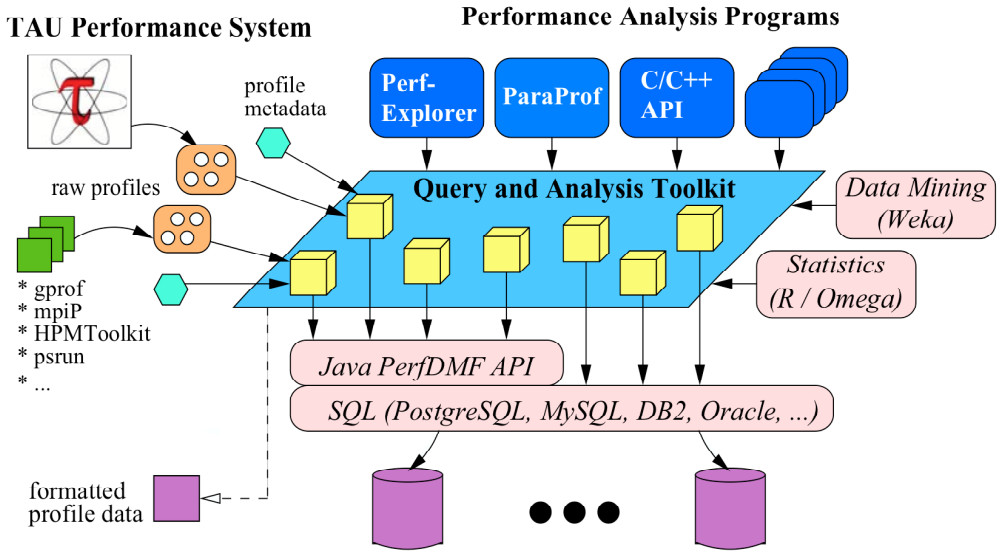

Figure 1. The PerfDMF architecture. PerfDMF includes an interface to a relational database to store profile data, an abstract profile query and analysis programming interface, and a toolkit of commonly used utilities for building and extending performance analysis tools.

4.1 PerfExplorer

Parallel applications running on high-end computer systems manifest a complexity of performance phenomena. Tools to observe parallel performance attempt to capture these phenomena in measurement datasets rich with information relating multiple performance metrics to execution dynamics and parameters specific to the application-system experiment. However, the potential size of datasets and the need to assimilate results from multiple experiments makes it a daunting challenge to not only process the information, but discover and understand performance insights. In order to perform analysis on these large collections of performance experiment data, we developed PerfExplorer [16], a framework for parallel performance data mining and knowledge discovery. The framework architecture enables the development and integration of data mining operations that will be applied to large-scale parallel performance profiles. PerfExplorer operates as a client- server system and is built on PerfDMF to access the parallel profiles and save its analysis results. The analysis is integrated with existing analysis toolkits (R , Weka), and provides for analysis extensions in those toolkits.

As shown in Figure 2, the architecture of PerfExplorer consists of two main components, the *PerfExplorer client* and the *PerfExplorer server*. The PerfExplorer Client is a standard Java client application, with a graphical user interface developed in Swing. The client application connects to the remote PerfExplorer server (also written in Java) using Remote Method Invocation (RMI), and makes processing requests of the server. The process of performing the data mining analysis is straightforward. Using the PerfDMF API, the server application makes calls to the performance database management system (DBMS) to get raw performance data. The server then passes the raw data to an analysis engine which performs the requested analysis. Once the analysis is complete, the PerfExplorer server saves the result data to the PerfDMF DBMS. Output graphics can also be requested at the server and images saved for later review. Because the analysis server is multi-threaded, it can continue to serve interactive requests to the client (or multiple clients) while performing background analysis tasks. The PerfExplorer client can even be launched from a web browser as a Java Web Start (JWS) application, providing an opportunity for collaborative analysis.

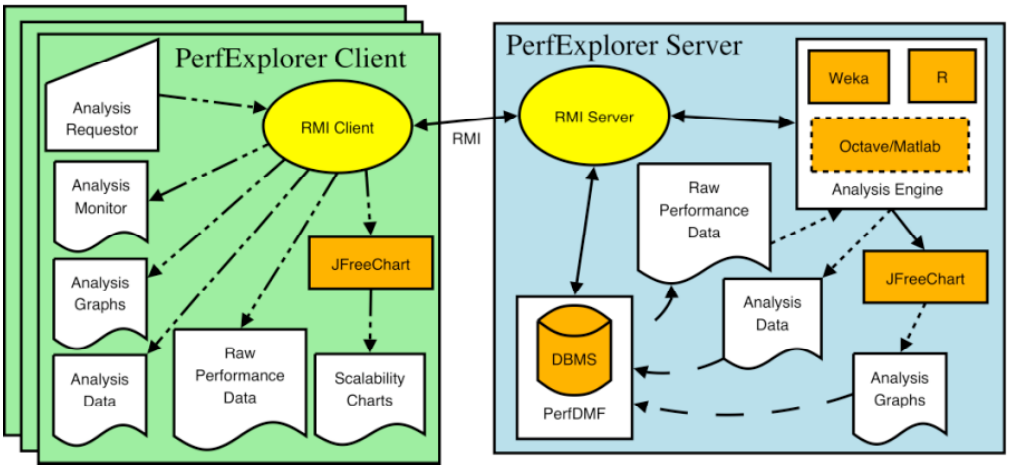

Figure 2. PerfExplorer architecture. Client users request data mining operations which are invoked on the server. Results are presented by the client. Multiple clients can be actively using the server.

There are a number of analysis methods available in PerfExplorer. When dealing with one trial of experiment data with hundreds or thousands of threads of execution, the user can request clustering of the data, which will aggregate the processes into representative groups and therefore simplify visual analysis of the data. Different clustering algorithms are implemented, such as *K*-means and hierarchical (nearest neighbor) clustering. In order to perform clustering, parallel performance profile data is organized into multi-dimensional vectors for analysis. Each vector represents one parallel thread or process of execution in the profile. Each dimension in the vector represents an event that was profiled in the application. Because clustering algorithms work best with ten or fewer dimensions, we have implemented several dimension reduction algorithms, including thresholds, random linear projection and Principle Components Analysis (PCA). Correlation analysis is also available, to evaluate the relationships between different profiled events in the application.

PerfExplorer provides a number of ways to examine the data distributions within large, high-dimensionality datasets, including a data summary table, box charts, histograms and normal probability plots. In addition, a 4-dimension correlation cube is available that selects the four most ``interesting'' events, and plots them as points with X,Y,Z and color intensity values. This type of view is useful in observing interactions between related events, such as communication barrier events and main calculation routines.

We have also developed several comparative analysis methods for multiple experiments or trials of parallel performance data. Parametric studies graphs can easily be generated by browsing to the profile data of interest, and selecting the type of chart. The charts available include total execution time, timesteps per second, relative efficiency/speedup, relative efficiency/speedup for one or all events, group percentage of total runtime (i.e. fraction of time spent in communication), a runtime breakdown of all events and correlating events with the total runtime. There are phase-based charts as well, which include relative efficiency/speedup per phase and the phase fraction of total runtime.

PerfExplorer has been applied to several CCA applications in the SciDAC CCTTSS project.

5. Online Performance Monitoring

The ability to interrogate component performance data during execution (i.e., performance monitoring) is supported by the measurement component through the query interface. This functionality is used, for instance, by the mastermind component for gathering performance information for performance model evaluation. However, the introspection TAU supports for CCA is local only to each process where the mastermind is executing. This is useful where performance decisions can be based on local performance data for components. In general, to support dynamic CQoS (a principal objective of CQoS activities in the

TASCS project), online performance monitoring is needed and both local and global performance views are required. TAU did not until recently provide a way to gather global performance data. Our work in the past year with TAUg [17] is described below.

5.1 TAUg

To enable a scalable parallel application to view its global performance state we designed *TAUg* as a portable runtime framework layered on the TAU parallel performance system [Huck2006]. TAUg leverages the MPI library to communicate between application processes, creating an abstraction of a global performance space from which profile views can be retrieved. Figure 3 portrays TAUg's system design.

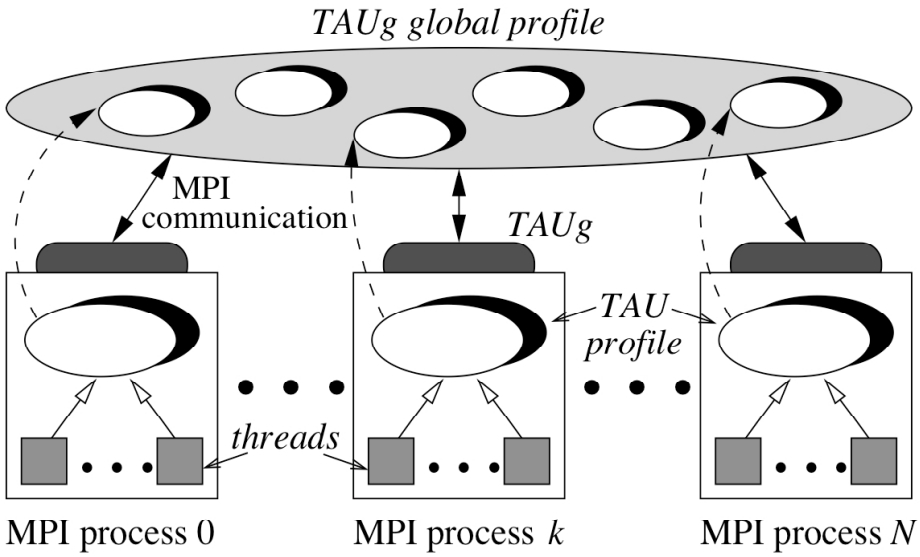

Figure 3. TAUg system design. MPI is used to establish global performance communicators which let application processes share performance information.

TAUg provides an API whose methods are designed in MPI style and callable from C, C++, and Fortran. Performance *view* and *communicator* definition and registration are supported to let the user create the global performance information services desired. The TAU_GET_VIEW() call then retrieves the performance view whose handle is supplied as an argument. A preliminary version of TAUg has been developed as a proof of concept and tested on Sweep3d and sPPM applications We are now extending TAUg for integration in the DOE SciDAC2 FACETS project.

6. Deployment in Component Frameworks and Applications

TAU has been applied to several component-based applications. Here, we summarize integration with CFRFS, ESMF, Uintah, and the SCIRun and SCIRun2 frameworks.

6.1 CFRFS

The TAU performance technologies have been used in the *Computational Facility for Reacting Flow Science (CFRFS)* [19] shock-hydro component-based simulation. Here, the proxy generator was used to produce mastermind monitoring proxies for each component. The mastermind component writes the component callgraph to disk when the simulation is complete. The tree pruning application then prunes the insignificant nodes based on the inclusive time spent in a component. The resulting graph will be an optimized core tree that identifies the major contributors to the component assembly's global performance. Any combination of pruned component instances can then be included to complete an approximately optimal global solution.

With the reduced callgraph, component families could be construct with multiple implementations. A global performance model is synthesized and evaluated by selecting an instance from each family and evaluating its individual model and its inter-component interactions. A complete, nearly optimal solution is achieved by adding in any implementation of the insignificant components that were pruned in the first step.

The CFRFS case study included a choice of two implementations, and the optimization phase correctly identified the implementation that provided the smallest execution time. However, the slower implementation is often the preferred choice by scientists because it provides better accuracy. This result indicates that a quality of service aspect (e.g., accuracy, robustness, etc.) is also important in evaluating an optimal selection. This case study is covered in [26].

6.2 ESMF

The *Earth System Modeling Framework (ESMF)* [11]is a component framework that can be used to build high performance, re-usable, numerical weather prediction, data assimilation, and other Earth science applications. Extensions to the core ESMF framework produced a component framework that interoperated with the CCA *Ccaffeine* framework. TAU's performance components were integrated in this dual framework and allowed us to evaluate the performance of climate components using an assembly of ESMF compliant components as shown in Figure 4.

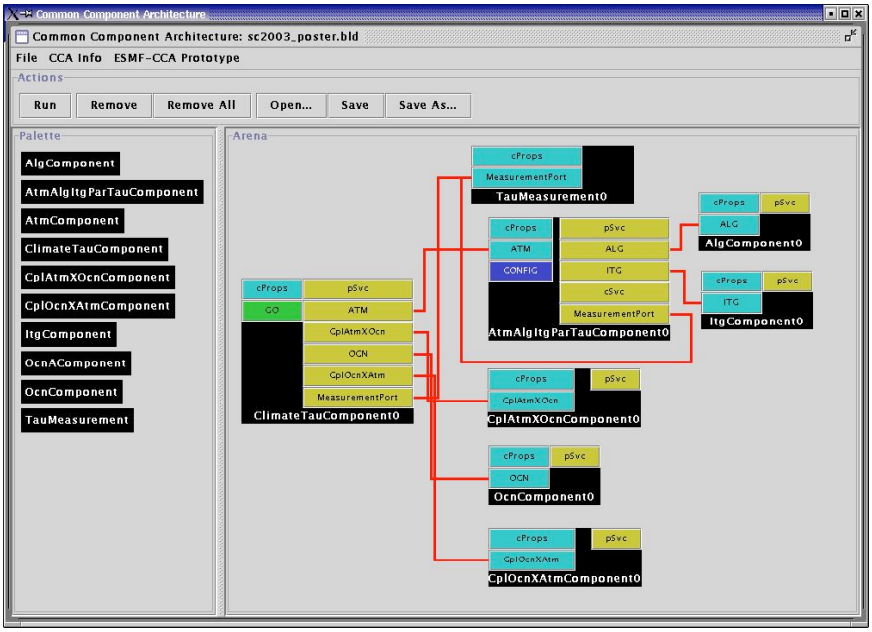

Figure 4. The ESMF component assembly.

TAU was also used to instrument ESMF framework automatically, providing both inter- and intra-component instrumentation for broader code coverage. To defer the choice of performance measurement (e.g., profiling, tracing, callpath profiling), TAU's measurement library was chosen by the combined framework at runtime. NOAA and NASA Codes such as SSI, GSI, and GEOSgcm use ESMF and TAU.

6.3 Uintah and SCIRun

The *Uintah Computational Framework (UCF)* [20] is the component-based software framework behind the University of Utah's C-SAFE project. It stems off from the SCIRun [21] architecture and provides for distributed computations (SCIRun only runs on shared memory systems). We worked directly with the Uintah developers to provide an integrated performance analysis framework within the Uintah build system.

The task scheduling subsystems are instrumented to dynamically create task level timers. Additionally, the entire source tree can be instrumented using PDT if the developer prefers more fine grained measurements.

We applied phase based profiling techniques to Uintah in a unique way. Rather than designated areas of code under phases, we instead defined the phases in Uintah as a data mapping. The spatial domain is divided into patches. Inside the scheduler, a given task is applied to a patch or set of patches. We instrumented this section to start a phase based on the patch or set of patches. In AMR simulations, we recorded both the AMR level and the patch identifier within that level. Figure 5 shows the phase profile for an AMR Uintah run.

Figure 5. Phase profile data for Uintah showing using AMR patch levels as phases.

Another scientific component framework developed at the University of Utah is the *SCIRun* Dataflow environment [21]. We worked with the core SCIRun developers to enable the use of TAU in the regular build system. In the SCIRun component framework, each module (component) is allocated a separate thread of execution. This makes differentiating components in the resulting profiles trivial. We need not insert proxy components or otherwise instrument the base framework.

6.4 SCIRun2

The *SCIRun2* framework, based on the CCA and SCIRun component frameworks, represents the next evolution in problem solving environments (PSEs) [22]. SCIRun2 supports distributed computing through distributed objects and parallel components are managed transparently over and MxN method invocation and data redistribution subsystem.

A group of SCIRun2 components, collectively referred to as *PERFume* was developed for performance analysis within the framework. In addition to quantifying the performance of the many numerical components, it boast the ability to quantify the overhead imposed by the component model abstraction of CCA. The PERFume components adhere to the CCA standards and should be operable in other CCA compliant frameworks. The components include performance monitoring, instrumentation, performance data storage, performance modeling and performance data visualization. The performance monitoring and instrumentation components are build using the TAU library.

7. Conclusion

The project has been very successful in meeting its objectives and working closely with the CCA community of framework and application developers. We hope to continue this interaction in the future.

References

1. U. Oregon, “TAU’s Performance Component,” URL: http://www.cs.uoregon.edu/research/paracomp/tau/cca , 2005
2. J. Ray, N. Trebon, S. Shende, R. C. Armstrong, and A. Malony, “Performance Measurement and Modeling of Component Applications in a High Performance Computing Environment: A Case Study,” *International Parallel and Distributed Processing Symposium (IPDPS’04)* , 2004. Also appears as Technical Report SAND2003-8631, Sandia National Laboratory, Sandia, CA.
3. N. Trebon, A. Morris, J. Ray, S. Shende, and A. Malony, “Performance Modeling of Component Assemblies with TAU,” *Concurrency and Computation: Practice and Experience*, CPE 1076, special issue of CompFrame 2005, John Wiley, 2006.
4. University of Oregon, “Program Database Toolkit,” URL: http://www.cs.uoregon.edu/research/paracomp/pdtoolkit/, 2004.
5. MFIX – Multiphase Flow with Interphase Exchanges URL: http://www.mfix.org , 2005.
6. Rutgers University, “The Accord programming framework: Enabling the development and management of autonomic applications,” URL: http://www.caip.rutgers.edu/~marialiu/Projects/Accord/index.htm, 2005.
7. K. Huck, A. Malony, R. Bell, and A. Morris, “Design and Implementation of a Parallel Performance Data Management Framework,” *International Conference on Parallel Processing (ICP’05),* 2005. (Awarded best paper.)
8. CCA Forum, “RPMS for CCA Components and Applications,” URL: http://www.cca-forum.org/~cca/.
9. University of Oregon, “Program Database Toolkit,” URL: http://www.cs.uoregon.edu/research/paracomp/pdtoolkit/ , 2004.
10. University of Oregon, “TAU’s Performance Component,” URL: http://www.cs.uoregon.edu/research/paracomp/tau/cca , 2004.
11. C. Hill, C. DeLuca, V. Balaji, M. Suarez, A. da Silva, and the ESMF Joint Specification Team, “The Architecture of the Earth System Modeling Framework*,” Computing in Science and Engineering,* Vol. 6, No. 1, January/February 2004. URL: http://www.esmf.ucar.edu
12. A. Malony, S. Shende, and R. Bell, “Online Performance Observation of Large-Scale Parallel Applications,” *Advances in Parallel Computing*, Vol. 13, special issue of Parco 2003 Symposium, Elsevier, B.V., pp. 761-768, 2004.
13. A. Malony, S. Shende, and A. Morris, “Phase-Based Parallel Performance Profiling," *Parallel Computing Conference (PARCO’05)*, 2005.
14. S. Shende, A. Malony, A. Morris, and P. Beckman, “Performance and Memory Evaluation Using TAU,” *Cray User's Group Conference (CUG 2006),* 2006.
15. University of Oregon, ``TAU's CCA Tools,'' URL: http://www.cs.uoregon.edu/research/tau/cca/, 2006.
16. K. Huck and A. Malony, “Perfexplorer: A Performance Data Mining Framework for Large-Scale Parallel Computing,” SC'05, Washington, DC, USA, 2005, IEEE Computer Society.
17. K. A. Huck, A. Malony, S. Shende, and A. Morris, “TAUg: Runtime Global Performance Data Access Using MPI,” *Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM-MPI 2006)*, LCNS 4192, pages 313--321, Bonn, Germany, 2006. Springer Berlin / Heidelberg.
18. 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), LCNS 3054, Springer, pp. 264-271, May 24-25, 2004.

19. H. Najm, et al. "A Computational Facility for Reacting Flow Science," SciDAC project, http://www.ca.sandia.gov/cfrfs/, 2006.
20. J. de St. Germain, J. McCorquodale, S. Parker, and C. Johnson, "Uintah: A Massively Parallel Problem Solving Environment," *IEEE International Symposium on High Performance and Distributed Computing*, IEEE, pp. 33-41. 2000.
21. S.G. Parker, "The SCIRun Problem Solving Environment and Computational Steering Software System," PhD thesis, University of Utah, 1999.
22. K. Zhang, K. Damevski, V. Venkatachalapathy, S. G. Parker, "SCIRun2: A CCA Framework for High Performance Computing,"*International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'04),* pp. 72-79, 2004.
23. A. Malony, S. Shende, N. Trebon, J. Ray, R. Armstrong, C. Rasmussen, and M. Sottile, "Performance Technology for Parallel and Distributed Component Software," *Concurrency and Computation: Practice and Experience*, Vol. 17, Issue 2-4, pp. 117-141, John Wiley, Feb-Apr, 2005.
24. S. Shende, A. Malony, C. Rasmussen, M. Sottile, "A Performance Interface for Component-Based Applications," *International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (IPDPS 2003),* IEEE Computer Society, 278, 2003.
25. N. Trebon, J. Ray, S. Shende, R.C. Armstrong, and A. Malony, "An Approximate Method for Optimizing HPC component Applications in the Presence of Multiple Component Implementations," Technical Report SAND 2003-8760C, Sandia National Laboratories, Livermore, CA, Dec. 2003. URL: http://infoserve.sandia.gov/sand_doc/2003/038760c.pdf.
26. N. Trebon, "Performance Measurement and Modeling of Component Applications in a High Performance Computing Environment," M.S. Thesis, University of Oregon, June 2005.
27. D. Bernholdt, et al., "A Component Architecture for High-Performance Scientific Computing," *International Journal of High Performance Computing Applications*, ACTS Collection Special Issue, SAGE Publications, 20(2):163-202, Summer 2006.