Making Performance Analysis and Tuning Part of the Software Development Cycle

Wyatt Spear, Sameer Shende, and Allen Malony
ParaTools, Inc., Eugene, OR
{wspear, sameer, malony}@paratools.com

Ricardo Portillo and Patricia J. Teller
University of Texas at El Paso
raportil@miners.utep.edu, pteller@utep.edu

David Cronk, Shirley Moore, and Dan Terpstra
University of Tennessee, Knoxville, TN
{cronk, shirley, terpstra}@cs.utk.edu

Abstract

Although there are a number of performance tools available to DoD users, the process of performance analysis and tuning has yet to become an integral part of the DoD software development cycle. Instead, performance analysis and tuning is the domain of a small number of experts who cannot possibly address all the codes that need attention. We believe the main reasons for this are a lack of knowledge about these tools, the real or perceived steep learning curve required to use them, and the absence of a centralized method that incorporates their use in the software development cycle.

This paper presents ongoing efforts to enable a larger number of DoD HPCMP users to benefit from available performance analysis tools by integrating them into the Eclipse Parallel Tools Platform (Eclipse/PTP), an integrated development environment for parallel programs.

1. Introduction

As evidenced by the CREATE[1] program, a Department of Defense (DoD) funded initiative to deploy computational engineering tool sets, improving the software development environment has been identified as an important task for the High Performance Computing Modernization Program (HPCMP)[2]. Although the process of code structuring and debugging should play the largest role during the software development cycle, code developers also should pay attention to increasing the performance of their applications, especially when designing codes destined to run on high-end computing systems. The growing gap between sustained performance and theoretical peak performance in HPC environments[3] only heightens the need for parallel code developers to devote more resources to performance tuning.

Although there are a number of performance tools available, they usually are employed by a small number of specialists who analyze and improve applications developed by third parties. This reduces the efficiency of performance tuning in two ways. First, relegating this task to a small number of performance specialists over-utilizes their expertise as there are usually more applications than they can address in a timely manner. Second, performance specialists often must devote time and effort to understanding the inner-workings of the target applications; this would not be necessary if application developers, who already have a thorough understanding of their own codes, took on the responsibility of performance analysis and tuning. Therefore, in general, enabling application developers to analyze and tune their own software increases the efficiency and employment of performance tuning.

We believe that the main reasons why developers do not utilize performance tools are a lack of knowledge about these tools, the real or perceived steep learning curve required to use them, and the absence of a centralized method that incorporates their use in the software development cycle. To address these issues, we investigated the inclusion of performance analysis tools within the Eclipse[4] Parallel Tools Platform[5] (Eclipse/PTP), an integrated development environment (IDE) for parallel programs. Within Eclipse/PTP, users can edit, build, run, debug, and analyze their parallel codes from an easy-to-use graphical user interface (GUI).

Our first phase in performance tool integration resulted in the implementation of plug-ins for several Eclipse IDE configurations[6], including Eclipse/PTP; these plug-ins integrated the functionality of the Tuning and Analysis Utilities (TAU)[7], a portable profiling and tracing toolkit for serial and parallel programs.

This paper describes new extensions to the Eclipse/PTP version of our TAU plug-in (from here on referred to as TAU PTP). The paper also describes the External Tools Framework (ETFw)[8][9], a performance tool integration method that evolved from our work with TAU PTP. ETFw is an XML-based framework that facilitates the integration, selection, and configuration of external performance tools within Eclipse/PTP. We demonstrated the capabilities of ETFw by integrating the functionality of several third-party performance tools into Eclipse/PTP, including Valgrind[10], SCALASCA[11], and VampirTrace[12].

We believe our integration strategy will facilitate the adoption of performance analysis tools by code developers for three reasons. First, integrating performance tools into a widely used IDE such as Eclipse increases their visibility. Second, introducing these tools into an environment with a GUI makes them more intuitive and reduces the effort required to learn how to use them. Third, centralizing these tools into one environment improves performance-tuning efficiency because developers do not need to switch back and forth between different performance analysis tool environments when working on the same source code.

The remainder of this paper is organized as follows: Section 2 explains the software stack that supports ETFw and TAU PTP. Section 3 introduces the TAU PTP plug-in and its initial functionality, while Section 4 describes the extensions we have made to TAU PTP. Section 5 introduces ETFw and Section 6 demonstrates its capabilities. Finally, Section 7 presents related work and Section 8 offers conclusions and future work.

2. Background

ETFw and the TAU PTP plug-in rely on an underlying hierarchy of software components, i.e., a software stack, which is depicted in Figure 1. Each component of the stack is described below, along with a definition of its role in support of ETFw and TAU PTP.

2.1 Eclipse

Serving as the foundation for our integrated environment, Eclipse is a widely used, open source, software development platform. By default, Eclipse serves as an IDE for Java programs but its most powerful feature, its plug-in framework[13], easily extends its capability to the support of the development and analysis of multiple programming languages and paradigms. This extensibility, coupled with its open source philosophy, has led to the creation of hundreds of plug-ins supported by a large community of dedicated contributors. Three of these third-party plug-ins, CDT, Photran, and PTP, serve as the next level of software components that support our TAU PTP plug-in, and enable the functionality of ETFw.

2.2 CDT and Photran Plug-ins

The majority of parallel codes are written in C/C++ and Fortran. Therefore, for our purposes, the C/C++ Development Tools (CDT)[14] and Photran[15] plug-ins are required to extend Eclipse and provide a fully functional IDE for C/C++ and Fortran, respectively. Both plug-ins provide a host of development features for their respective languages such as project management, code debugging, automated builds, syntax coloring, and code completion. Users can download and install CDT into their existing Eclipse environment or they can acquire an official release of Eclipse with CDT already installed. Photran, originally a separate plug-in project, recently has been merged into the larger PTP project, which is described in the following section. This merger is expected to grow Photran's contributor base and produce more frequent and consistent plug-in releases.

2.3 Parallel Tools Platform (PTP) Plug-in

The Parallel Tools Platform plug-in is extending Eclipse to support high-performance parallel computing for programs written in Fortran, C, and C++, with parallelism implemented using MPI[16] and/or OpenMP[17]. Therefore, the CDT and, optionally, Photran plug-ins are required to support these paradigms. The idea is to provide a single interface for editing, building (i.e., compiling and linking), and executing parallel applications. PTP supports Open MPI[18] and MPICH2[19] MPI implementations as well as various runtime environments, including batch and remote job submission configurations. In addition, PTP provides a series of static analysis tools for parallel programs in the form of the Parallel Language Development Tools (PLDT) plug-in. These tools enable developers to perform several code analysis tasks, including deadlock detection via MPI barrier analysis and OpenMP thread concurrency analysis.

2.4 Third-Party Performance Analysis Tools

Taking advantage of the Eclipse plug-in framework, and the above-described plug-ins, we created our own plug-in, TAU PTP, and, subsequently, ETFw to facilitate the integration of third-party performance tools. A brief description of the tools we integrated, with the help of TAU PTP and ETFw, follows.

TAU: The Tuning and Analysis Utilities – TAU[7], is a profiling, tracing, and visualization toolkit for performance analysis of serial and parallel programs. TAU also can interact with several third-party performance tools, including PAPI, SCALASCA, and Vampir, to perform multi-tool performance analysis tasks.

PAPI: The Performance Application Programming Interface (PAPI)[20] provides a set of library functions that, when called within an application, can access hardware performance counters. Its main goal is to provide a performance counter interface that is portable and consistent across disparate hardware architectures.

SCALASCA: The SCalable performance Analysis of LArge SCale Applications (SCALASCA) tool[11], a profiling and tracing toolkit that is an extension of KOJAK[21], can analyze separate trace files in parallel, thereby increasing the feasibility of event trace analysis of large parallel program executions.

Valgrind: Operating as an instrumentation framework, Valgrind[10] can automatically detect many memory management and threading errors. It is comprised of six tools that enable analysis tasks such as memory leak detection and branch-prediction profiling.

VampirTrace: The VampirTrace[12] library enables users to instrument their parallel codes and generate traces of MPI communication events during program execution. These traces can then be fed into Vampir, a visualization and analysis tool for MPI traces.

3. TAU PTP Plug-in

As mentioned previously, TAU is a performance toolkit that can generate, analyze, profile and trace information from parallel program executions. Users who wish to capture and analyze performance data with TAU must perform four basic steps:

1) **Code Instrumentation:** TAU captures performance data via the use of Application Programming Interface (API) library calls inserted within the target program. By default, TAU instruments the entry and exit points of all routines within an application. TAU also provides the ability to define specific source code regions to instrument. Users can do this manually by inserting TAU API calls at strategic locations in the target program or employ TAU's Performance Database Toolkit (PDT) to selectively instrument the code automatically.
2) **Configure TAU:** TAU supports a rich set of data collection options for profiling and tracing. Users must define these options at compile time either in their application's makefile or at the command line.
3) **Compile and Link:** Once the source code is instrumented and a TAU configuration is defined, the target program must be compiled and linked to TAU's API library. TAU provides a set of compile scripts that automate these tasks as well.
4) **Execute and Analyze:** During the execution of the target program, TAU's API calls capture the desired information and store it in a specified directory or performance database. Users can then analyze these data using TAU's pprof or ParaProf tools. TAU also can generate these performance data in formats compatible with several third-party trace and profile analysis tools, including KOJAK and Valgrind.

Although TAU provides utilities such as PDT and compiler scripts, which facilitate the process of acquiring and analyzing performance data, it is apparent that users still must memorize, or refer to, a large set of instrumentation and configuration task work flows to analyze their codes. Thus, a method is required that enables users to employ TAU's functionality both intuitively and with a minimum amount of effort.

The core functionality of our TAU PTP plug-in[6] takes advantage of Eclipse's open framework environment to provide a graphical interface for the configuration and execution of TAU profiling and tracing tasks. As seen in Figure 2, TAU PTP lists available TAU configuration options as a series of tabs and checkboxes. Once all desired options are selected, users can profile or trace their codes under this configuration with a single menu or button selection. We believe this approach greatly improves TAU's usability because it allows users to forgo memorization of data collection and instrumentation options and automates the compilation, linking, and execution of a TAU instrumented program; all within a portable and widely available GUI.

Since our initial implementation of TAU PTP, we added functionality that further integrates TAU data collection and analysis tasks. We also generalized our tool integration method and developed ETFw to facilitate the integration of additional performance tools into Eclipse/PTP. Both of these endeavors are presented in the following sections.

4. TAU PTP Plug-in Extensions

This section describes new functionality that we added to our original TAU PTP plug-in implementation.

4.1 Selective Instrumentation

Our initial implementation of TAU PTP utilized TAU's default instrumentation scheme, which captures performance data at the entry and exit points of all user routines in the target program. Although this usually is appropriate for the initial stages of performance analysis and identification of performance bottlenecks, instrumenting all routines may result in unacceptable runtime overhead.

To address this issue, we integrated TAU's selective instrumentation functionality into TAU PTP. We added three basic instrumentation schemes:

1) **Specification-based Instrumentation:** Users may create a selective instrumentation file that specifies source code regions, routines, or events of interest, and directs TAU PTP to instrument the program according to this specification.
2) **Highlight-based Instrumentation:** Users may use the mouse to highlight specific regions of code and automatically generate a selective instrumentation file to place start and stop TAU API calls at the beginning and end of these regions.
3) **Event-based Instrumentation:** Users may select a project, source file, or routine, and direct TAU PTP to capture specific runtime event information from these regions. As can be seen in Figure 3, TAU PTP can automatically instrument a target program to capture various types of runtime events, including memory allocation/de-allocation instructions, I/O calls, and static/dynamic callpath phases.

In addition to these new instrumentation capabilities, users now can direct TAU PTP to create a binary executable of the instrumented target application, which can be launched from outside the Eclipse IDE. This feature enables users to instrument their codes and launch them in non-interactive execution environments such as batch job submission systems, which are employed in many time-shared HPC installations.

4.2 Management of Performance Data

TAU PTP initially stored all generated performance data in a local directory. Although acceptable for small analysis studies, a directory-based storage approach does not scale in usability or performance as the number and size of performance data increases. As in other data management domains, the use of a database can greatly increase the manageability and accessibility of large performance data sets.

TAU supports the use of performance databases with its Performance Data Management Framework (PerfDMF), an API/toolkit that sits on top of a Database Management System (DBMS). PerfDMF's primary function is to link TAU to a third-party DBMS such as MySQL or Oracle. Once linked, TAU stores all generated performance data in a specified database supported by the underlying DBMS.

Our TAU PTP plug-in now supports performance databases via the use of TAU's PerfDMF toolkit. Once a database is specified, TAU PTP uses it to store data generated during profiling and tracing tasks. TAU PTP also makes performance database entries visible and accessible via a profile viewer integrated into the Eclipse/PTP environment. We also enhanced the accessibility and shareability of performance data by enabling users to employ databases that reside remotely over a network. We believe these enhancements greatly simplify the analysis of large performance datasets such as those generated by parallel program profiles and traces.

4.3 Visualization of Performance Data

TAU's main analysis tool, ParaProf, is a portable and scalable performance utility for profile data visualization; see Figure 4. It is compatible with various profile specifications including TAU, gprof, and KOJAK data formats. When using ParaProf, users may choose to load specific data files for analysis or link ParaProf to a performance database created via TAU's PerfDMF utility, described in Section 4.2. ParaProf provides several data categorization methods, including views at the thread, MPI, and hardware counter levels, and can derive new metrics from user-designated profile events. ParaProf also provides various formats for 2D and 3D visualization.

To further integrate TAU into Eclipse/PTP, we included ParaProf's functionality into our TAU PTP plug-in. Now users can select a data set from a profile viewer within Eclipse/PTP and launch ParaProf to visualize it. This enhancement, in conjunction with the additional functionality described in previous sections, enables users to perform all four steps required for TAU-based analysis without leaving the Eclipse/PTP environment. Specifically, users now can instrument their codes, select TAU data collection options, compile and link instrumented code, and perform visualization analysis on the generated performance data all within a single integrated development environment.

4.4 Memory Leak Detection

As mentioned in Section 4.1, TAU PTP can instrument target programs automatically to capture memory allocation/deallocation instruction events during a program's execution. In addition to enabling this type of memory profiling, we also integrated TAU's memory leak detection functionality into the TAU PTP plug-in.

Users who wish to perform memory leak detection with TAU PTP must first instrument their codes to capture memory events. As already mentioned, TAU PTP can do this automatically at the project, source file, and routine levels. When automatic memory instrumentation is activated, TAU PTP analyzes the source code of the program and, for Fortran programs, inserts memory instrumentation calls at allocate and deallocate statements. For C and C++ programs, *malloc* and *free* wrappers are used to redirect calls to allocation and deallocation routines. When the target program executes, memory instrumentation is activated, which triggers atomic events that track the volume of memory used. Context events map the executing callstack to the memory allocation, deallocation, and leak events. This allows the user to observe the source file, line, and variable names associated with a memory leak in the source code along a callpath. Once this profile information is generated, ParaProf can take these data and render various views showing when and where memory leaks occurred during a parallel program's execution.

4.5 PAPI Hardware Counter Support

Most modern day processors contain special-purpose registers that can count events such as floating-point operations and cache misses at runtime. Since these counters reside in hardware, and are accessible to the user, performance counters are widely used as a low-overhead approach to high-granularity code profiling. To access these counters, users must utilize an architecture-specific API. Unfortunately, this forces users to instrument their codes differently for each unique target architecture, thus, making performance analysis across computing platforms a nontrivial task. PAPI addresses this issue by providing a single hardware counter API that is portable across a wide range of processor architectures.

We enhanced the usability of PAPI by integrating its functionality into our TAU PTP plug-in. With the use of PAPI, TAU PTP now enables the automatic identification and selection of available hardware counters, see Figure 5, as well as the automatic instrumentation of PAPI API calls within Eclipse/PTP. TAU PTP also can work with Component PAPI (PAPI-C), a PAPI extension that allows users to configure support for non-CPU events such as temperature, kernel statistics, and Ethernet and Myrinet interface events. Our plug-in enables users to capture performance counter data for both profiles and traces. For performance counter profiles, users can view the generated data within the ParaProf visualization tool, which, as described in Section 4.3, is integrated into TAU PTP.

4.6 Parametric Study Support

Parametric studies, where machine/application pairs are run under various compile and runtime parameter sets, are widely used in the area of performance analysis. These studies are particularly useful for identifying scalability bottlenecks in the software and hardware stack of HPC environments. Thus, such studies enable program developers to tune their codes to reduce performance penalties from such bottlenecks. Additionally, parametric studies aid the design of future HPC systems by informing designers on which parts of the system software and hardware to focus improvement.

Given the importance of parametric studies in HPC tuning, we added support for these analyses within TAU PTP[22]. As can be seen in Figure 6, now users may use a configuration interface specific to parametric analysis to define sets of parameter values. Users may choose from various parameter types, including number of processors, compiler optimization levels, and environment variables. Once defined, TAU PTP runs the target application with each parameter set in succession and stores the generated data in a performance database. Once the data are generated, TAU's PerfExplorer tool can be used to analyze and visualize the performance data within Eclipse/PTP. PerfExplorer is a multi-experiment analysis and data-mining tool that was designed to provide parametric study analysis and intelligent analysis of results using performance data, metadata, analysis scripts, and inference rules.

4.7 Two-stage Routine and Loop Level Profiling

Performance analysis typically begins with routine-level instrumentation, which can guide further instrumentation to routines that use significant resources. For example, scientific applications are typically iterative in nature and use multiple nested loops and, thus, use significant computational resources. In the second stage of performance analysis, a user may instrument the code at loop boundaries to track the performance of loops. Two-stage routine and loop level profiling can enhance a user's understanding of which loops in the application contribute most to the overall runtime and the system resources consumed by them.

As described in Section 4.1, TAU PTP enables users to instrument their codes at many levels of the program hierarchy, this includes routine and loop-level instrumentation. Thus, users may use the TAU PTP plug-in to realize TAU-specific two-stage routine and loop-level profiling. The TAU performance tool also can insert instrumentation calls from other profiling tracing tools including VampirTrace. We have enabled TAU PTP to instrument target programs with VampirTrace calls as well, thus, providing users with additional two-stage analysis options.

VampirTrace provides a performance measurement library for generating time-stamped MPI event traces from parallel program executions. Our TAU PTP plugin now supports automatic insertion of VampirTrace trace functions at selective routine and loop exit/entry locations. Once the target program is instrumented, a user can automatically link the instrumented program with the appropriate VampirTrace libraries, execute the program, capture the generated time-stamped events, and, as shown in Figure 7, visualize the data with the Vampir analysis tool.

5. External Tools Framework (ETFw)

During the development of the TAU plug-ins, which included TAU PTP, it became evident that much of the work being done was applicable to other performance analysis systems and similar command-line based tools. At a high level, such tools typically operate on some combination of compilation, execution, and analysis steps and their inputs are similar to those of TAU.

To take advantage of this congruity, the workflow logic and User Interface (UI) elements, which were initially hard-coded into the original TAU plug-ins, were converted to a generalized API. Additionally, to make the system more easily accessible and extensible, we developed an XML interface for defining both performance tool workflows and their UIs within Eclipse/PTP. The result is the general-purpose External Tools Framework (ETFw). ETFw allows both tool and application developers to integrate performance analysis systems into an Eclipse environment without the effort and expertise that are required to develop new Eclipse plug-ins. In fact, XML workflow definitions for external performance tools can be added or updated without restarting the Eclipse platform.

Although ETFw generalized much of the hard-coded behavior of the original TAU plug-ins, advanced TAU-specific functionality remains encapsulated within a plug-in structure, see Sections 3 and 4. However, the advanced API extension points used by the TAU-specific plug-ins are available to other tools that require logic or UI elements that are too application-specific for the ETFw to handle.

The ETFw's XML workflow format, depicted in Figure 8, consists of three fundamental elements, which define the *compilation*, *execution*, and *analysis* steps of the workflow. The order, number, and presence of these steps may vary depending on the intent of the workflow and the employed analysis tools.

- The *compilation step* assigns compiler commands to be used for the relevant programming languages.
- The *execution step* defines commands to be composed with the target executable, if any. This covers tools such as Valgrind that take the target application as an input argument.
- The *analysis step* defines a series of commands that may be run on any data generated during program execution.

Each application or tool defined in an XML workflow may have its command and input parameters specified in the XML file. Alternatively, command-line options may be specified, which will appear in the Eclipse/PTP UI, where the user may enable, disable, and assign values to them dynamically. Once an XML workflow has been composed, it can be modified easily to suit different use cases. It also can be distributed to other users, who can easily load it into their Eclipse/PTP environments and run their applications with the performance analysis workflow, without concerning themselves with tool invocation details.

In addition to adding support for arbitrary performance tools, the ETFw's abstraction of performance tool operations simplifies the implementation of more complicated workflows. This includes workflows that require multiple executions of the target application, such as parametric studies[22]. ETFw is now part of the PTP plug-in and is, thus, available to all users with a current Eclipse/PTP IDE configuration.

6. ETFw Case Studies

This section describes the new analysis capabilities that we added to Eclipse/PTP by integrating additional performance tools via our ETFw implementation. The ETFw tool definitions required to run these performance tools within Eclipse/PTP were released as part of the TAU PTP plug-in distribution available from the official TAU website[7]. As shown in Figure 9, users can select from a list of integrated performance tools once the appropriate XML tool definitions are loaded into Eclipse/PTP.

6.1 PerfSuite and VampirTrace/Vampir Support

To demonstrate the usability of our integration framework, we created ETFw XML tool definitions for PerfSuite and VampirTrace/Vampir tool workflows. Figure 8 presents an abridged version of our tool definition for VampirTrace and Vampir. As this figure demonstrates, VampirTrace compile, execute, and Vampir analysis commands can be easily described in our ETFw XML format. Users who load this tool definition into their Eclipse/PTP environment will be presented with additional UI options that enable the specified VampirTrace/Vampir functionality. The tool definitions we created to enable VampirTrace, Vampir and PerfSuite functionality within Eclipse/PTP are available along with our distribution of TAU PTP.

6.2 Parallel Event-Trace Analysis with SCALASCA

Event-trace analysis is a useful, low-overhead method for identifying in-depth runtime program behavior. During a program's execution, time-stamped events such as function calls and messages are captured and stored in a trace file. Once the target program finishes, the resulting trace file can be thoroughly analyzed for performance bottlenecks and other behavioral characterizations. Unfortunately, the growing size and complexity of modern programs is resulting in equally large, and unmanageable, trace file data sets. This is particularly apparent in parallel program analysis where it is not uncommon to run thousands, if not hundreds of thousands, of processes at a time.

SCALASCA, an extension of the KOJAK profiling and tracing toolkit, addresses this issue by optimizing the analysis procedure for trace files generated from parallel program executions. Instead of aggregating all data into a single trace file and analyzing it sequentially, SCALASCA optimizes this process by generating smaller, local trace files on each compute node and analyzes each in parallel.

Since SCALASCA's tracing functionality is geared towards HPC performance analysis, we integrated this tool into Eclipse/PTP using ETFw's XML tool definition functionality. With our SCALASCA tool definition, we enabled users to instrument, trace, analyze, and visualize their target codes with SCALASCA within Eclipse/PTP. Additionally, as shown in Figure 10, we developed interoperability between our TAU PTP plug-in and SCALASCA by enabling users to visualize TAU PTP-generated data with SCALASCA and vice versa, thereby providing a flexible analysis environment for these tools within Eclipse/PTP.

6.3 Memory Leak Detection with Valgrind

As demonstrated in Section 4.4, TAU PTP now provides memory leak detection to users of Eclipse/PTP. Although TAU PTP's method of leak detection is sound, it is sometimes prudent to characterize memory behavior via other detection methods, which may confirm or supplement the original analysis. To complement TAU PTP's leak detection functionality, we created an ETFw XML tool definition for Valgrind's Memcheck utility.

Valgrind's method of leak detection differs from that used by TAU PTP. While TAU PTP runs an instrumented target program natively and analyzes the generated performance data offline, Memcheck emulates an un-instrumented target in a virtual machine and detects memory leaks upon program exit. These two methods can serve to confirm each other's results.

Our ETFw tool definition for Valgrind provides a simple process for memory leak detection. When the user selects Valgrind from a configuration menu and profiles the target application, Valgrind spawns the program under a virtual machine and tracks all memory events. As shown in Figure 11, Valgrind's analysis results are then displayed in Eclipse/PTP's console tab upon program termination.

7. Related Work

There are a few examples of other performance analysis plug-ins that are available for the Eclipse IDE platform. The Test & Performance Tools Platform (TPTP)[23] provides many performance tool capabilities, including execution, memory, and thread behavior analyses. Although TPTP is a robust, well-supported performance analysis tool for Eclipse, currently it can only analyze Java applications. Although Java is a widely used programming language, parallel programs are mainly written in C/C++ and Fortran and, therefore, are unable to take advantage of TPTP's functionality.

However, there are two performance tools that do support C/C++ and Fortran, i.e., Intel's Vtune Performance Analyzer[24] and IBM's Visual Performance Analyzer (VPA)[25]. Both tools provide a wide variety of mature profiling and tracing capabilities. Although these tools enable application developers to analyze the performance of parallel codes within Eclipse, they do not provide the extensibility of TAU PTP. TAU PTP and ETFw allow easy integration of further performance analysis tools into Eclipse/PTP, thus, providing a highly moldable and adaptive interface for current and future analysis technologies.

8. Conclusions

As evidenced by the CREATE program, improving the software development environment has been identified as an important task by the HPCMP. As HPC systems scale up to hundreds of thousands of processors, the growing gap between theoretical performance and sustained performance provides a major incentive for application and system developers to devote more time and effort into performance analysis and tuning of parallel codes. Although mature and robust performance tools are freely available to the programming community, the use of these tools is largely the domain of a minority of performance analysis experts. To increase the awareness, usability, and efficiency of available performance tools, we developed a performance tool integration methodology for the Eclipse/PTP integrated development environment. We demonstrated the power of this methodology by integrating a large set of TAU analysis capabilities into Eclipse/PTP. Additionally, we demonstrated the flexibility of our implementation by providing case studies of other analysis tools that were easily added to Eclipse/PTP via our ETFw integration framework.

With regards to future work, currently we are focusing on enabling TAU PTP to analyze target programs that reside on remote file systems. This capability will greatly facilitate performance analysis of parallel programs since many HPC environments often are utilized through a remote connection. In addition, we plan to take advantage of Eclipse/PTP's new remote development features to reduce response time lags inherent in running the Eclipse IDE over a network, which is currently the case for many users of Eclipse/PTP.

TAU PTP has been released in the public domain distribution of TAU, which is available from the official website[5]. Users also can acquire TAU PTP and ETFw as part of the PTP plug-in, which is freely available to the Eclipse community via the PTP website[5].

We believe the integration of performance tools into Eclipse/PTP will increase their use by parallel program developers, thus, enabling them to produce applications that take full advantage of increasingly powerful HPC systems.

Acknowledgments

This publication was made possible through support provided by DoD HPCMP User Productivity Enhancement and Technology Transfer (PET) activities through Mississippi State University under contract No. GS04T01BFC0060. The opinions expressed herein are those of the author(s) and do not necessarily reflect the views of the DoD or Mississippi State University.

References

[1] S. Arevalo, et al., “A New DoD Initiative: the Computational Research and Engineering Acquisition Tools and Environments (CREATE) Program,” *Journal of Physics: Conference Series*, vol. 125, pp. 012090, August 2008.

[2] U.S. Department of Defense, “High Performance Computing and Modernization Program,” available online at: http://www.hpcmo.hpc.mil [Accessed: Apr 21, 2009].

[3] W. Kramer, “How Terascale Experience Will Shape Petascale Systems,” presented at *27th Asia-Pacific Advanced Network Meeting*, Kaohsiung, Taiwan, March 2009, available online at: http://www2.jp.apan.net/meetings/kaohsiung2009/presentations/opening/kramer.pdf [Accessed: April 21, 2009].

[4] Eclipse Foundation, “Eclipse Integrated Development Environment,” available online at: http://www.eclipse.org [Accessed: April 21, 2009].

[5] Eclipse PTP Project, “Eclipse Parallel Tools Platform,” available online at: http://www.eclipse.org/ptp [Accessed: April 21, 2009].

[6] W. Spear, A. Malony, A. Morris, and S. Shende, “Integrating TAU with Eclipse: a Performance Analysis System in an Integrated Development Environment,” in *Proceedings of the 2nd International Conference on High Performance Computing and Communications (HPCC’06)*, *Lecture Notes in Computer Science*, vol. 4208/2006, pp. 230-239, September 2006.

[7] University of Oregon, “TAU - Tuning and Analysis Utilities,” available online at: http://www.cs.uoregon.edu/research/tau [Accessed: Apr 21, 2009].

[8] W. Spear, A. Malony, A. Morris, and S. Shende, “Performance Tool Workflows,” in *Proceedings of the 8th International Conference on Computational Science (ICCS’08), Lecture Notes in Computer Science*, vol. 5103/2008, pp. 276-285, July 2008.

[9] Eclipse PTP Project, “ETFw - External Tools Framework,” available online at: http://wiki.eclipse.org/PTP/ETFw/PTP_External_Tools_Framework [Accessed: April 21, 2009].

[10] Valgrind Project, “Valgrind Tool Suite,” available online at: http://valgrind.org [Accessed: April 21, 2009].

[11] Jülich Supercomputing Centre, “SCALASCA Toolset,” available online at: http://www.fz-juelich.de/jsc/scalasca [Accessed: April 21, 2009].

[12] Technische Universität Dresden, “VampirTrace Library,” available online at: http://www.tu-dresden.de/zih/vampirtrace [Accessed: April 21, 2009].

[13] J. Amsden, “Levels of Integration: Five Ways You Can Integrate the Eclipse Platform,” available online at: http://www.eclipse.org/articles/Article-Levels-Of-Integration/levels-of-integration.html [Accessed: April 21, 2009].

[14] Eclipse CDT Project, “Eclipse C/C++ Development Tooling - CDT,” available online at: http://www.eclipse.org/cdt [Accessed: April 21, 2009].

[15] Photran Project, “Photran - an Integrated Development Environment for Fortran,” available online at: http://www.eclipse.org/photran [Accessed: April 21, 2009].

[16] Message Passing Interface Forum, “MPI: a Message Passing Interface Standard,” in *International Journal of Supercomputer Applications*, vol. 8, number 3/4, pp. 165-414, Fall-Winter 1994.

[17] L. Dagum and R. Menon, “OpenMP: An Industry-Standard API for Shared-Memory Programming,” in *Computing in Science and Engineering*, vol. 5, issue 1, pp. 46-55, January-March 1998.

[18] Open MPI Project, “Open MPI: Open Source High Performance Computing,” available online at: http://www.open-mpi.org [Accessed: April 21, 2009].

[19] Argonne National Laboratory, “MPICH2: High Performance and Widely Portable MPI,” available online at: http://www.mcs.anl.gov/research/projects/mpich2 [Accessed: April 21, 2009].

[20] University of Tennessee, “PAPI - Performance Application Programming Interface,” available online at: http://icl.cs.utk.edu/papi [Accessed: April 21, 2009].

[21] University of Tennessee, “KOJAK - Kit for Objective Judgment and Knowledge-based Detection of Performance Bottlenecks,” available online at: http://icl.cs.utk.edu/kojak [Accessed: April 21, 2009].

[22] K. Huck, W. Spear, A. Malony, S. Shende, and A. Morris, “Parametric Studies in Eclipse with TAU and PerfExplorer,” in *Proceedings of the Workshop on Productivity and Performance (PROPER’08), Euro-Par 2008 Workshops - Parallel Processing,* vol. 5415/2009, pp. 283-294, 2009.

[23] Eclipse TPTP Project, “Eclipse Test & Performance Tools Platform - TPTP,” available online at: http://www.eclipse.org/tptp [Accessed: April 21, 2009].

[24] Intel Corporation, “Vtune Performance Analyzer.” Available online at: http://www.eclipse.org/tptp [Accessed: Apr 21, 2009].

[25] IBM Corporation, “Visual Performance Analyzer.” Available online at: http://www.alphaworks.ibm.com/tech/vpa [Accessed: Apr 21, 2009].

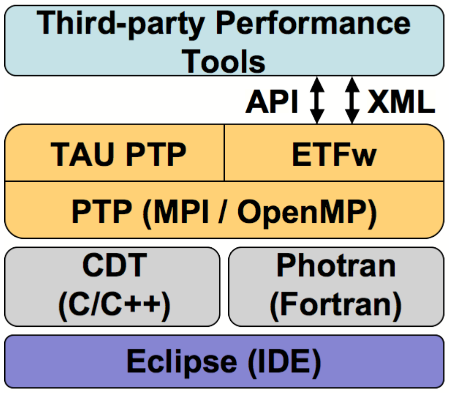


Figure 1. TAU PTP and ETFw Software Hierarchy

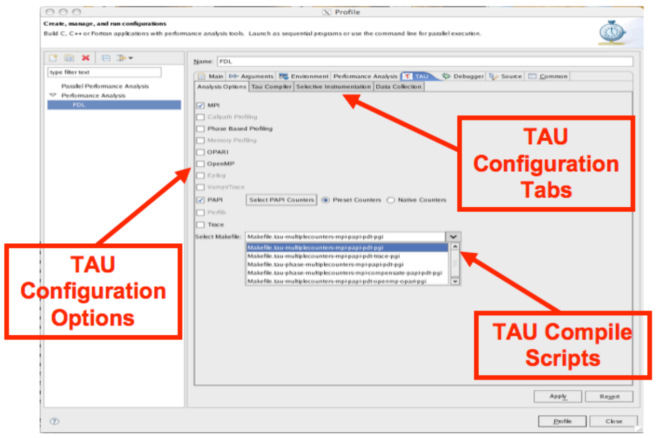


Figure 2. TAU PTP enables easy selection of TAU profiling and tracing options

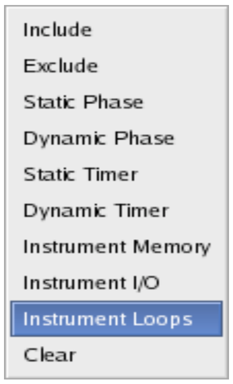


Figure 3. TAU PTP Selective Instrumentation context menu

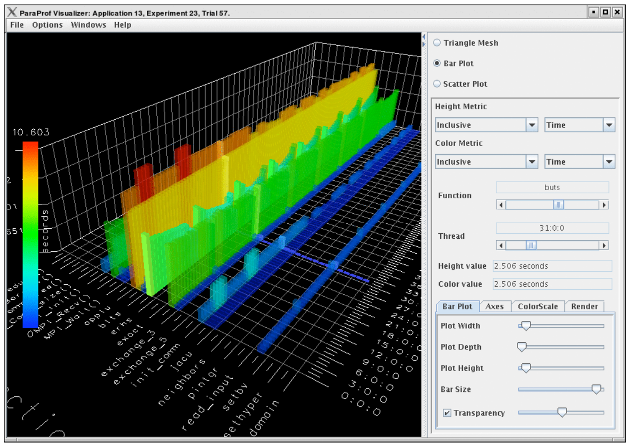
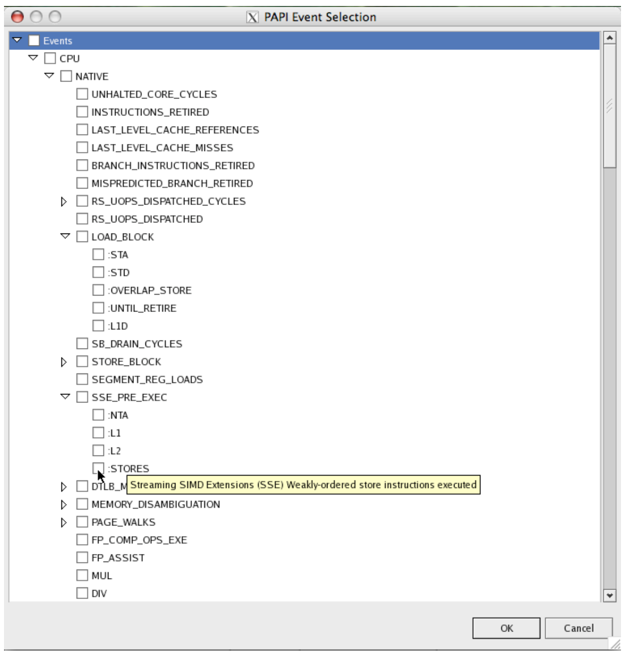


Figure 4. The ParaProf visualization tool


PAPI Event Selection

Events
- CPU
  - NATIVE
    - UNHALTED_CORE_CYCLES
    - INSTRUCTIONS_RETIRED
    - LAST_LEVEL_CACHE_REFERENCES
    - LAST_LEVEL_CACHE_MISSES
    - BRANCH_INSTRUCTIONS_RETIRED
    - MISPREDICTED_BRANCH_RETIRED
    - RS_UOPS_DISPATCHED_CYCLES
    - RS_UOPS_DISPATCHED
    - LOAD_BLOCK
      - :STA
      - :STD
      - :OVERLAP_STORE
      - :UNTIL_RETIRE
      - :L1D
    - SB_DRAIN_CYCLES
    - STORE_BLOCK
    - SEGMENT_REG_LOADS
    - SSE_PRE_EXEC
      - :NTA
      - :L1
      - :L2
      - :STORES
    - DTLB_M Streaming SIMD Extensions (SSE) Weakly-ordered store instructions executed
    - MEMORY_DISAMBIGUATION
    - PAGE_WALKS
    - FP_COMP_OPS_EXE
    - FP_ASSIST
    - MUL
    - DIV

OK Cancel

Figure 5. Native x86_64 events show hierarchical events in the Eclipse/PTP interface.

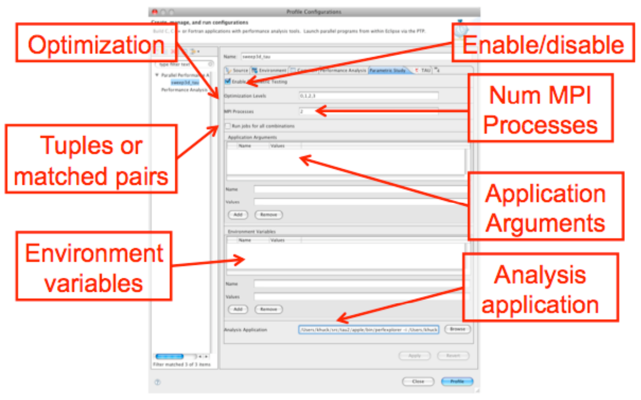


Figure 6. TAU PTP Parametric Study configuration menu

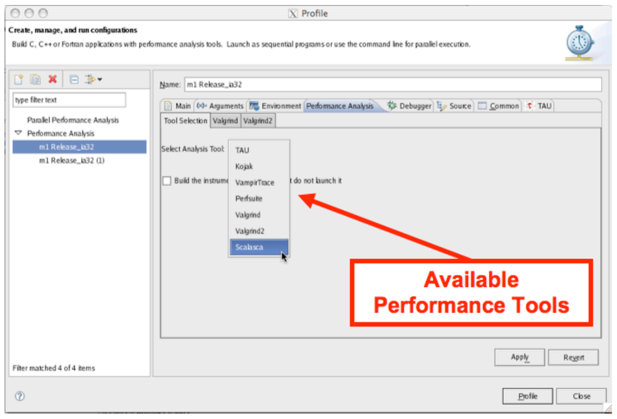


Figure 9. Selection of integrated performance tools via ETFw

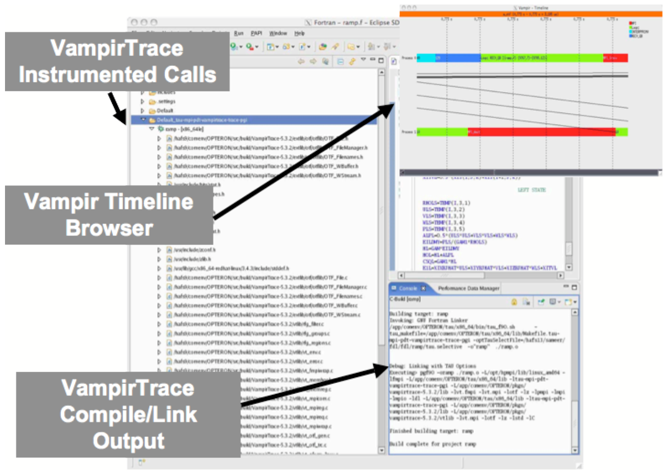

VampirTrace Instrumented Calls
Vampir Timeline Browser
VampirTrace Compile/Link Output

Figure 7. VampirTrace intrumentation and Vampir visualization of trace data within Eclipse/PTP

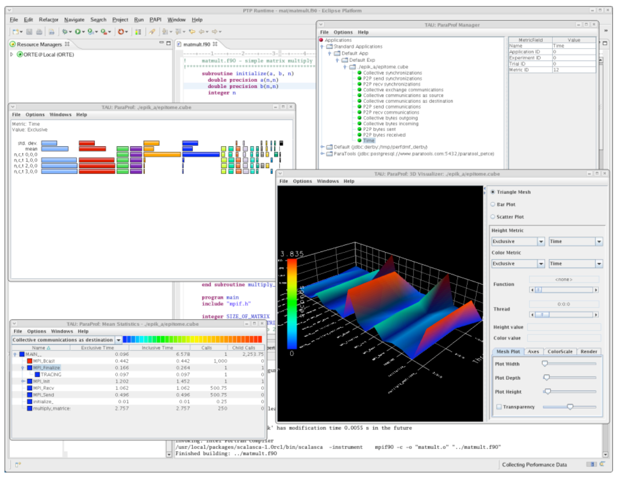


Figure 10. TAU PTP plug-in demonstrating the ParaProf and SCALASCA performance analysis tools

```
<toolset.01>
  <tool name="Vampirtrace">
    <compile>
        <CC command="vtcc" group="vampirtrace">
            <argument value="-vt:cc"/>
        </CC>
    </compile>

    <execute>
        <utility command="mpirun" group="mpi">
            <argument value="-np 4"/>
        </utility>
    </execute>

    <analyze>
        <utility type="view" command="vampir" group="vampir">
            <argument value="a.otf"/>
        </utility>
    </analyze>
  </tool>
</toolset.01>
```

Figure 8. Sample ETFw XML file for VampirTrace (color coded for clarity)

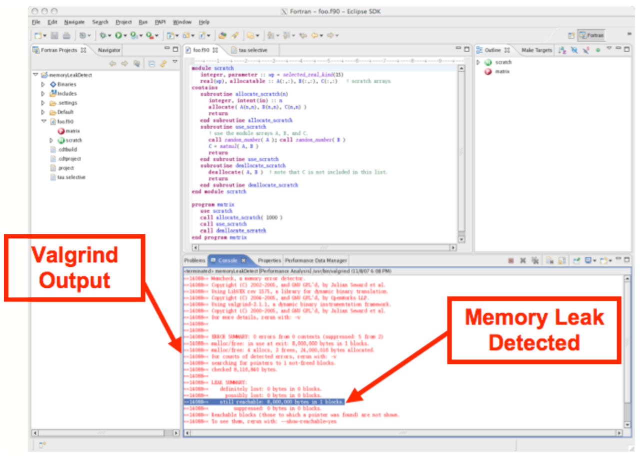


Figure 11. Valgrind memory leak detection within Eclipse/PTP