Performance Tool Workflows

Wyatt Spear, Allen Malony, Alan Morris, and Sameer Shende

Performance Research Laboritory
Department of Computer and Information Science
University of Oregon, Eugene OR 97403, USA
{wspear,malony,amorris,sameer}@cs.uoregon.edu

Abstract. Using the Eclipse platform we have provided a centralized resource and unified user interface for the encapsulation of existing command-line based performance analysis tools. In this paper we describe the user-definable tool workflow system provided by this performance framework. We discuss the framework's implementation and the rationale for its design. A use case featuring the TAU performance analysis system demonstrates the utility of the workflow system with respect to conventional performance analysis procedures.

1 Introduction

Performance analysis is an important component of software development, especially in high performance and parallel computing. With the proliferation of multicore systems and the growing reliance upon parallel computing in science and industry, the number of programmers who need to analyze and optimize application performance as a matter of course is likely to increase.

Collecting performance data can be a complicated and time consuming undertaking. Depending on the performance metrics one intends to collect and the tools employed the steps may include source level instrumentation, compilation with performance tool specific compilers or options, execution with performance tool specific options or composed with a data collection tool, data collection, storage, analysis, format conversion and visualization. It often requires knowledge of several distinct tools to effectively perform even a subset of these tasks. This must be accomplished and any technical hurdles must be overcome before the true goal of performance analysis, actually using collected performance data to improve the efficiency of an application, can be pursued.

Because performance analysis tools are usually command-line based, multi-step performance analysis procedures are generally either done by hand or performed with scripts. Such scripts are invariably specific to the tools being used and sometimes even the application being analyzed. In any case, managing performance tool inter-operation is left to the end user. Expertise in the use of individual tools and the collective use of multiple tools must be developed individually, or obtained via documentation rather than any easily deployable programmatic means.

In more conventional venues of software development the Integrated Development Environment has found favor for its productivity enhancing features. Few

M. Bubak et al. (Eds.): ICCS 2008, Part III, LNCS 5103, pp. 276–285, 2008.

IDEs offer significant support for the development of high performance applications. Where IDEs do offer performance analysis solutions, they are typically unique to the IDE in question. If other tools are required the user must return to the command line or resort to manual data manipulation.

The performance analysis framework for Eclipse attempts to address these difficulties by providing an extensible, modular, general system for defining performance analysis workflows. Workflow systems have been shown to increase efficiency and ease of use for complex computational activities composed of discrete steps [6]. With the growing complexity of and need for performance analysis, the benefits of workflow techniques seem quite applicable. The expertise required to perform a given performance analysis task or series of tasks can be encapsulated in a workflow definition for easy distribution and deployment. So long as the necessary tools are available on the system, the user need only select the desired workflow and set any necessary starting parameters.

The performance analysis framework we have developed offers a modular, extensible solution to the problem of performance analysis in IDEs by encapsulating existing command line based tools. In addition to the basic requirement of offering performance analysis tool functionality, it allows tools to be linked together in a workflow of performance analysis steps. The components comprising this system and the steps that led to its development are described below.

2 Eclipse

The Eclipse integrated development environment [3] began as a platform for the development of Java applications. In recent years its functionality has expanded to support multiple languages and programming paradigms. The C/C++ Development Tools (CDT) [2] and Photran [14] projects provide functionality for C/C++ and Fortran development, respectively. The Parallel Tools Platform (PTP) [7] project extends the capabilities of the CDT and Photran by offering parallel development, launch and debugging. These tools make Eclipse a promising means of enhancing traditional command line tools with IDE based development techniques in high performance computing. However, the requirements of high performance computing extend beyond the productivity-improving features of a standard IDE.

The tools available for performance analysis and the breadth of features that they cover make development of new tools specifically for a given IDE a difficult task. This is further complicated by the fact that most parallel application developers are already accustomed to some set of tools and development procedures available on the command line. Those who take the first steps toward IDE based parallel application development may find the productivity gains provided by the IDE significantly curtailed if they are faced with unfamiliar or incomplete tools. The solution to this problem is to wrap the IDE around existing tools, fully exposing their capabilities in the GUI. By making performance tools available within Eclipse we not only simplify their usage, often the tools incorporated into the environment are made more usable than in their initial command-line based incarnations.

The performance analysis framework required a set of general, modular components that could be combined to allow any command-line based performance analysis tool to be incorporated. The movement of other projects toward supporting parallel application development on the Eclipse platform and the platform's modularity and extensibility makes it an ideal focus for this work.

3 TAU

The performance analysis framework was initially created to add support for the TAU (Tuning and Analysis Utilities) [8][10]. Like many other performance analysis systems TAU is primarily oriented toward usage on the command line. It supports numerous platforms and parallel programming paradigms including MPI and OpenMP. Its data collection options include tracing, profiling, call-path profiling, hardware counter data collection and inter-operation with various other performance analysis systems.

The performance data produced by TAU can be broadly classified as either trace output or profile output. Depending on TAU's configuration one or both types of data may be generated in any of several formats. TAU provides several performance analysis tools, including ParaProf [1], a scalable graphical profile analysis tool. It also provides support for database management of performance profiles via the PerfDMF [4] database system.

TAU's flexibility necessitates some complexity in deployment. A new configuration must be created for each combination of performance analysis options to be used. In most cases the application to be analyzed must be instrumented with TAU API calls and linked with the appropriate TAU library.

The use of TAU is greatly simplified by the use of the Performance Database Toolkit (PDT) [5] and TAU's compiler wrapper scripts. PDT enables automatic parsing and instrumentation of a project's source code, using selective instrumentation if necessary. TAU's compiler wrapper scripts automatically include the necessary invocations of PDT and modifications to the compilation command to include TAU libraries. Collecting performance data can be as simple as replacing the default compiler command with the TAU compiler script in a project's makefile and specifying the desired compiler and environmental options. However, once the raw performance data has been collected there are still numerous paths to be taken in manipulation and analysis of that data.

4 The TAU Plug-In

The most fundamental way to use TAU in conjunction with Eclipse is to replace the default compilers used by Eclipse's build system with the TAU compiler scripts. Although this can be done manually, the numerous options and features available to TAU at compile time must still be known to the user to be invoked. Furthermore the process of adjusting analysis parameters and rebuilding the application when different performance analysis data are desired still necessitates manual string editing. Manually collecting performance data with TAU in Eclipse

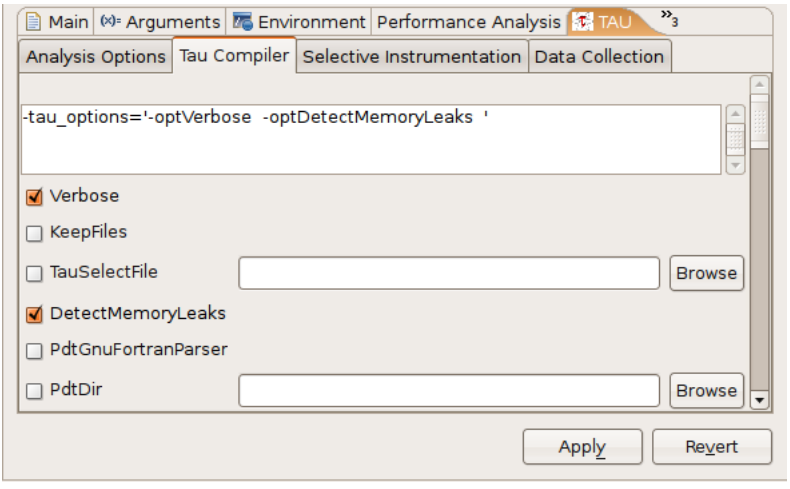

Fig. 1. TAU Compiler Option Selection

is no more difficult than typical command-line based usage. The same generally applies to other performance tools that one might attempt to deploy in Eclipse. There is clear room for improvement.

The core of the TAU plug-in for Eclipse [9] allows the user to select a TAU configuration graphically. The visual exposure and online documentation of TAU's performance data collection options and the easy selection of those options within the Eclipse environment (Fig. 1) makes the plug-in a significant improvement over manual inclusion of TAU in the build configuration.

The TAU plug-in also provides options for automatically storing generated profile data and viewing that data in the ParaProf profile analysis tool. In effect, it provides a single, static workflow using a pre-set series of tools.

5 The Performance Framework

The development of the TAU plug-in and subsequent refactorings and optimizations resulted in a modular set of routines for the invocation of tools at various stages of the Eclipse build and launch procedures. It also resulted in the creation of a dynamic UI generator which can read user specified option definitions and create a corresponding set of options in Eclipse's user interface. Once defined, these options are automatically applied to the relevant build or launch steps.

Though these components had been directed toward the use of TAU in Eclipse, many of their capabilities are just as easily applied to other performance analysis tools. The TAU specific elements of the core plug-ins amounted to the strings defining the compiler and option names and the logic specific to TAU configuration and option selection. Rather than require other tool developers to develop their own Eclipse plug-ins, we determined to extend the TAU plug-ins to support

essentially arbitrary series of commands which could be assembled for any given tool without significant knowledge of Eclipse plug-in development.

Converting the plug-in to a system capable of dynamic command specification was not a trivial undertaking. The parameters of a selected tool workflow needed to be associated with data structures that could efficiently propagate through each stage of performance data generation and analysis. Furthermore, the internal representations of tool specific configurations had to support essentially arbitrary combinations of commands and parameters. Fortunately, the highly modular implementation of the performance framework makes it fairly simple to add new functionality or extend the workflow definition format should an analysis tool with an unsupported interface be discovered.

6 Performance Analysis Workflow

There are two essential goals of the workflow definition format. The first is to grant flexibility to tool developers to integrate their tools into Eclipse without the need to modify their tools or to devote a great deal of time to learning Eclipse plug-in development. The second is to grant performance tool users the ability to easily configure and deploy project-specific performance analysis capabilities within Eclipse with a relatively small investment of time and effort.

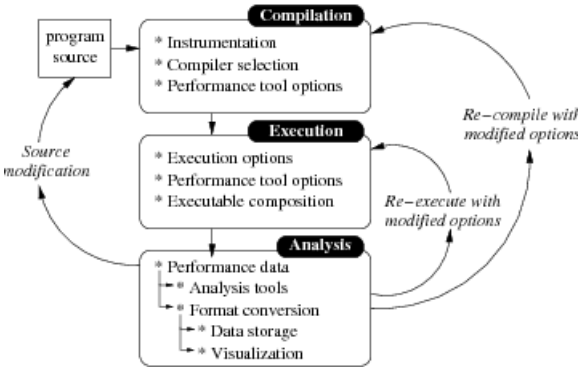

Fig. 2. A General Performance Analysis Workflow

The performance workflows are defined in an XML document stored outside of the Eclipse workspace. This document may contain an arbitrary number of individual workflow definitions. Workflow definitions are broken down into compilation, execution and analysis phases (Fig. 2). A single workflow may consist of several tools. By composing performance data collection and analysis tools it is possible to produce an instrumented program, execute that program, collect the performance data and display that data in the chosen performance data viewer with a single mouse click.

6.1 Compilation

The compilation phase definition specifies compilers and compiler arguments to be used by the Eclipse build system. This is primarily useful for performance tools that require recompilation of the program for instrumentation. Typically this section will include specification of a compiler, often a compiler wrapper script associated with the performance tool, and any relevant arguments. The arguments specified in the tool definition will be combined with any arguments provided by the build configuration selected for the build process.

Workflows that define a compilation phase require the user to select a build configuration rather than an existing executable file in Eclipse's launch configuration. The selected build configuration us used as the basis for the construction of the new executable. The user interface adjusts for the selection of an executable or a build configuration depending on the requirements of the selected workflow.

The default behavior of the performance plug-ins, when recompilation is defined in the workflow, is to compile and execute the program and then run any specified analysis tools once data generation is complete. However the execution phase may be disabled, for example if the user intends to run the compiled program outside of Eclipse.

6.2 Execution

The execution phase accommodates the use of tools that prepend the project's compiled executable. This element was intended to support performance tools such as perfsuite's psrun [13]. However, it has also proven useful for composing applications for more general purposes. The ability to define a tool configuration that automatically runs a compiled application with mpirun, for example, has proven useful on systems where the PTP is unavailable. The composition of multiple applications in the launch phase is supported. A typical use of this capability is to initiate a parallel launch of an application using mpirun composed with an analysis tool.

Unlike most of the other workflow components, the modifications to the standard launch system for the execution phase to support arbitrary tools were almost entirely independent of the work done to support TAU. TAU itself does not make use of composed executable launching. Additionally, unlike modifying the build configuration system, manually arranging a composed executable launch in Eclipse is not a straightforward process.

6.3 Analysis

The analysis phase is initiated after execution. It consists of a series of commands and their arguments to be run in sequence. Information on the output of the workflow's performance tool is used to pass performance data to the specified post-processing, data-management or visualization applications. Additionally, the output of one application can be directed as an argument to another. The flexibility of the tool definitions available in the analysis phase allow a great deal of creativity in its use by tool developers and users.

It is difficult to dynamically capture the output of the executed program for use in a given analysis-phase application. In some cases the names of generated performance files can not be predicted. Presently it is necessary specify the output to be accessed in the tool definition on a per-application basis. Eventually it should be possible to specify this dynamically from within the launch configuration.

7 Use Cases

The large number of performance tools and the varied capabilities of each tool presents a vast number of combinations which can be difficult to wade through for those who simply want to collect performance data and optimize their software without focusing on the technical details of performance analysis. Even when the procedure is straightforward, undertaking a multi-step performance analysis workflow manually can be time consuming. The use cases presented here provide some insight into the utility of performance analysis workflows.

7.1 TAU

A common use case for the TAU performance analysis system calls for generation of trace and profile data for a given application. The trace data is then merged, converted into a trace format associated with a particular trace viewer and analyzed in that trace viewer. The profile data is stored in a database and viewed in ParaProf.

The individual steps are relatively simple once one becomes familiar with the various tools involved. However, even with expertise in the various tools, the procedure is still time consuming. To get from compilation to data visualization as rapidly as possible, it is necessary to observe each of the individual steps and initiate the next as soon as the previous is complete.

A workflow defined in the performance analysis framework makes this procedure much more straightforward. After selecting the relevant TAU workflow (Fig. 3) it is still necessary to select a TAU configuration and specify any compilation or execution specific options. However the user interface provided for these options require only introductory knowledge of TAU's capabilities.

With the relevant options selected the performance workflow may be launched. The application will be recompiled with TAU's compiler wrapper scripts, automatically generating instrumented source and linking it with the relevant libraries. It bears note that for a tool lacking automatic instrumentation capabilities instrumentation may still need to be performed manually.

The TAU-instrumented executable will be launched immediately with any parameters specified in the Eclipse launch configuration, in addition to those provided by the TAU configuration interface. Depending on the type of TAU configuration specified trace and/or profile data may be produced along with any output normally generated by the program being analyzed.

By default, profile data will be uploaded to a user-specified PerfDMF database if one is available. The ParaProf profile analysis tool will be launched on the profile data stored within the database. The profile files are deleted if the database

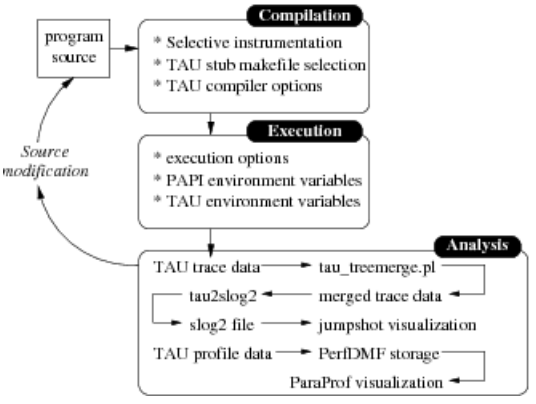

Fig. 3. A Workflow for a Common TAU Use Case

upload is successful, otherwise they are placed in a local directory identified by the launch configuration name and timestamp. The logic for these operations is coded directly into the performance framework. This behavior is left from the original TAU specific plug-in. However a similar series of analysis steps could easily be specified using the workflow definition system.

Trace data manipulation was not supported directly in the TAU specific iteration of the plug-in. In this workflow trace management consists of the following steps. First tau_treemerge.pl is called to merge tracefiles from multiple processes. tau2slog2 is then called on the merged trace output to produce a .slog2 file. Finally, to display the trace Jumpshot is called on the .slog2 file.

Once a launch configuration is configured with the desired workflow and the relevant performance collection and application options are specified it will persist within the Eclipse workspace. Changes to the source code or the launch parameters can be tested immediately with minimal oversight by the user.

7.2 Valgrind

A relatively simple workflow using a tool quite different from TAU involves the use of the memory analysis tool valgrind [12]. A valgrind workflow requires only the specification of the valgrind executable in the execution phase, along with any desired options.

When this workflow is selected in the launch configuration no other modifications are necessary. The executable of the selected project will be composed with valgrind and any options defined in the workflow. The launch will then proceed normally. The output from valgrind will be displayed in Eclipse's console view unless different output behavior for valgrind is specified in the workflow.

8 Future Work

As of this writing, some of the performance framework's more advanced capabilities are still somewhat limited to supporting TAU. One priority is extending the

selective instrumentation capabilities of the system to support the various selective instrumentation schemes of alternative performance analysis tools. A related goal is to support the selection tool options usable by multiple tools, such as PAPI hardware counters [11], for tools other than TAU using a single interface.

Currently the analysis tools are only invoked after an execution phase has completed. Although this covers the most typical use cases, it may be useful to initiate an analysis workflow on performance data already present on the filesystem. Support for analysis-only operations is a priority, though the interface for this feature is not likely to be an extension of the launch configuration.

Presently the workflow definition format only allows for linear workflows using some combination of compilation, execution and analysis commands. Soon, the modular nature of the workflow components will allow the addition of some logical elements to the workflow system. For example, rather than terminate at the end of the analysis phase, data collected in analysis could be evaluated and the compilation and execution phases repeated with modified parameters until a certain set of conditions is met. This will be useful for performance analysis procedures that require iterative data collection. The initial implementation of this capability will likely be to facilitate performance scalability testing.

Ultimately we hope to provide a means of visually defining and modifying tool workflows within Eclipse. This will remove the need for users or developers to interact with the tool definition XML file and make the full capabilities of the workflow system more accessible and obvious.

9 Conclusion

By providing a general framework for performance analysis in a popular IDE we hope to simplify the process of performance analysis just as IDEs have assisted with simplifying other aspects of the software development cycle. Ideally this work can benefit not only existing software developers, but will also be of use to newcomers to high performance software engineering who may be more accustomed to IDE-based software development.

Perhaps more importantly, we have created a means for expert users to encapsulate their expertise in a programmatic way. A given series of performance analysis operations can be defined as a workflow by a tool developer or advanced user. This can then be made available to others who desire the final output of the potentially complex performance analysis procedure, but have no desire to engage in the manual, multi-step process every time they need to collect the data from their application.

References

1. Bell, R., Malony, A.D., Shende, S.: A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis. In: Proc. EUROPAR 2003 conference. LNCS, vol. 2790, pp. 17–26. Springer, Heidelberg (2003)
2. CDT - C/C++ Development Tools, `http://www.eclipse.org/cdt`

3. Eclipse, http://www.eclipse.org
4. Huck, K., Malony, A.D., Bell, R., Li, L., Morris, A.: PerfDMF: Design and implementation of a parallel performance data management framework. In: Proc. International Conference on Parallel Processing (ICPP 2005). IEEE Computer Society, Los Alamitos (2005)
5. Lindlan, K.A., Cuny, J., Malony, A.D., Shende, S., Mohr, B., Rivenburgh, R., Rasmussen, C.: A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates. In: Proceedings of SC 2000: High Performance Networking and Computing Conference, Dallas (November 2000)
6. Oinn, T., et al.: Taverna/myGrid: aligning a workflow system with the life sciences community. In: Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M. (eds.) Workflows for e-Science: scientific workflows for Grids, pp. 300–319. Springer, Guildford (2007)
7. PTP - Parallel Tools Platform, http://www.eclipse.org/ptp
8. Shende, S., Malony, A.D.: The TAU Parallel Performance System. International Journal of High Performance Computing Applications, ACTS Collection Special Issue (2005)
9. Spear, W., et al.: Integrating TAU With Eclipse: A Performance Analysis System in an Integrated Development Environment. In: Gerndt, M., Kranzlmüller, D. (eds.) HPCC 2006. LNCS, vol. 4208, pp. 230–239. Springer, Heidelberg (2006)
10. TAU - Tuning and Analysis Utilities, http://www.cs.uoregon.edu/research/tau/home.php
11. Moore, S., Cronk, D., Wolf, F., Purkayastha, A., Teller, P., Araiza, R., Aguilera, M., Nava, J.: Performance Profiling and Analysis of DoD Applications using PAPI and TAU. In: Proceedings of DoD HPCMP UGC 2005. IEEE, Nashville, TN (2005)
12. Nethercote, N., Seward, J.: Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. In: Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007), San Diego, California, USA (June 2007)
13. Kufrin, R.:PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux. In: 6th International Conference on Linux Clusters: The HPC Revolution 2005. Chapel Hill, NC (April 2005)
14. Photran - Fortran Development Tools, http://www.eclipse.org/photran/