Integrating TAU With Eclipse: A Performance Analysis System in an Integrated Development Environment

Wyatt Spear, Allen Malony, Alan Morris, Sameer Shende

{wspear, malony, amorris, sameer}@cs.uoregon.edu

Abstract. The Eclipse platform offers Integrated Development Environment support for a diverse and growing array of programming applications and languages. There is an increasing call for programming tools to support various development tasks from within Eclipse. This includes tools for testing and analyzing program performance. We describe the high-level synthesis of the Eclipse platform with the TAU parallel performance analysis system. By leveraging Eclipse's modularity and extensibility with TAU's robust automated performance analysis mechanisms we produce an integrated, GUI controlled performance analysis system for Java, C/C++ and High Performance Computing development within Eclipse.

1 Introduction

IDEs (Integrated Development Environments) are increasing in popularity across many venues of software development. At the same time they are encompassing a larger set of roles within the software development process, such as advanced debugging, version control and software deployment. Many projects focus on developing new software tools to fill the growing requirements of IDE software systems. However, integration of preexisting command-line or otherwise standalone development tools into an IDE environment also offers several advantages. These include interoperability with preexisting data and tools that support the newly integrated components, user familiarity and, in many cases, the advantages derived from starting with a more mature, robust system.

The Eclipse platform supports a popular Java IDE with additional support available for for C and C++[0]. Support for Fortran and other languages is in development. High performance application development within Eclipse is also receiving increased attention[0]. However, performance analysis is an important component of the software development process that has received little integrated support in any of the Eclipse platform's IDEs, outside of the scope of Java development.

A number of specialized performance analysis systems exist. Although Eclipse projects may be able to make use of them, their interfaces are external to the platform. Most are command line based and not designed with IDE integration in mind. The complexity of a full, multi-language, performance analysis system makes implementing one specifically for a given IDE generally impractical. The difficulty is compounded further by issues of architectural and operating system support. Some performance analysis features, such as hardware event recording, may require platform specific configuration which would complicate deployment of any performance analysis solution internal to the IDE. A practical solution to these problems is to allow the IDE to wrap a consistent interface around an extant performance analysis system already configured and deployed on the desired platform.

TAU(Tuning and Analysis Utilities)[0] is a comprehensive performance analysis system which supports a wide array of architectures, operating systems and programming languages. Its features include automation of many performance analysis operations and utilities for converting its performance data output to a number of formats commonly used by other performance analysis tools. This makes it ideal for integration with the diversely deployed, extensible Eclipse platform.

The following two sections discuss Eclipse and TAU respectively, focusing on the features that facilitate and benefit from their integration. This is followed by a description of the Eclipse plugins that achieve the integration with TAU and an explanation of their implementation and future goals. Finally, the conclusion summarizes the success of the initial work.

2 Eclipse

Eclipse[0] is a popular software platform with support for customized IDE functionality. Its default set of plugins is designed for Java development but the Eclipse community is providing increasing support for other languages such as C/C++ and Fortran. Recently, support for high performance computing has also been provided via the Parallel Tools Platform(PTP)[0].

Two distinct advantages of the Eclipse platform are its portability and extensibility. The former is provided largely by Eclipse's Java-based implementation. Because it is implemented in Java it can be run consistently on Windows, Macintosh and many Unix based OSes. This can facilitate development of software for multiple architectures and helps provide a consistent interface if a user needs to migrate between different systems or operating systems during a project's development.

Because Eclipse is open source users are free to modify and extend its functionality as they see fit. This freedom is enhanced by Eclipse's fundamentally modular architecture. The result is a diverse array of enhancements and plugins made to increase Eclipse's functionality for software development, as well as other tasks.

2.1 The Eclipse Platform

Eclipse is essentially a platform for the support of plugins. The core of Eclipse provides the basic user interface and internal control mechanisms but virtually every useful activity that can be performed within Eclipse relies on a plugin. Typically a single application, such as the Java IDE, is comprised of a set of plugins subdivided according to their individual functions. Eclipse's plugin API allows the inclusion of functionality from simple context menus to advanced program build managers.

Although Eclipse's default set of plugins are aimed at Java development they provide a significant level of general functionality and support for the expansion of its capabilities. The basic file controls, text editors and preferences widgets, for example, are plugin components that can be reused in many other applications.

Eclipse provides automated utilities for plugin installation and upgrading, but the manual procedure is as simple as moving the folder or jar file associated with a given plugin to the appropriate directory in the Eclipse installation. Additionally Eclipse allows control over which plugins are instantiated at runtime.

2.2 Eclipse Plugins

The Eclipse platform's facilities for extension and modification have resulted in a rich and growing supply of plugins for a wide variety of purposes. These serve both within Eclipse's original capacity as an IDE platform and in other more diverse applications. Performance analysis tools are by no means absent from this. The Test and Performance Tools Platform (TPTP)[0] is an example of a significant effort toward a fully integrated performance analysis system for Eclipse. Although the TPTP possesses a deliberately extensible API, presently it primarily supports analysis only within the JDT.

There are three Eclipse plugin collections, or projects, that lend themselves directly to the integration of the TAU performance analysis tools. The Java Development Tools (JDT), the C/C++ Development Tools (CDT) and the Parallel Tools Platform (PTP) all facilitate the development of programs and the use of programming paradigms that are supported by TAU and none include their own internal mechanisms for performance analysis.

JDT The JDT[0] assists with Java development by providing a context sensitive source editor, project management and development control facilities, among other features. Most relevant to TAU integration is the JDT's run configuration management system. This grants control over the main method executed when a project is run from within Eclipse along with program arguments, environment variables and related options.

Because the JDT is a fundamental component of the default Eclipse package, it is the most mature of the projects to be integrated with TAU. Its API is fairly static and well documented, easing the development of plugins that add to its functionality.

CDT Many of the CDT's features are comparable to those of the JDT. In particular the source editing features provide similar functionality such as syntax based colorization, search, replacement and refactoring tools and tree views of source structure. However, the build system of the CDT is naturally quite different. It supports both the use of external makefiles and an internally constructed "Managed" makefile system. In either case the compilation and linking of programs within the CDT is accomplished via user specified compilers and compiler options.

The run management systems of the JDT and CDT are similar in their user interfaces and internal APIs. In the case of a CDT, the run management system requires specification of a compiled binary file.

PTP The PTP has made significant advances in support of parallel program development within Eclipse. While it is possible to program and compile parallel programs from within the CDT, it has no innate facilities to launch the resulting executables with a given parallel runtime. The PTP grants the ability to both launch and debug parallel programs from within Eclipse. Presently, the PTP only supports the OpenMPI[0] MPI implementation. However, parallel launching and debugging capabilities support for other MPI implementations and OpenMP are in development.

In many ways, the PTP acts as an extension of the CDT. This is especially evident in its compilation and execution system APIs. Thus, the creation of plugins that make use of these components is quite similar between the two projects.

3 TAU

TAU is a mature performance analysis system designed to operate on many different platforms, operating systems and programming languages. In addition to collecting a wide range of performance data it includes resources for performance data analysis and conversion to third party data formats.

Many of TAU's functions are closely bound to the underlying architecture of the system where the analysis takes place. Therefore, TAU is generally configured and compiled by the user to create custom libraries for use in performance analysis. In addition to generating system specific libraries, this configuration process allows specification of many performance analysis options allowing an extremely diverse range of performance experiments to be carried out with TAU. Each separate configuration operation produces a stub makefile and a library file that is used to compile an instrumented program for analysis.

3.1 Instrumentation

TAU's fundamental functionality is based on source code instrumentation. At the most basic level this consists of registering the entry and exit of methods within the program via calls to the performance analysis system. Performance analysis of a given program can be focused on a given set of functions or phases of the program's execution by adjusting which functions are instrumented. A common application of such selective instrumentation is to exclude small, frequently called routines to help reduce performance analysis overhead.

Manual instrumentation can be time consuming, especially for large programs. TAU includes utilities to perform automatic instrumentation of source code. These utilities rely on the Program Database Toolkit, or PDT[0], to analyze the code so proper instrumentation points can be programmatically determined.

TAU also provides compiler scripts which act as wrappers of the compilers described at TAU's configuration. Use of these scripts in place of a conventional compiler results in fully instrumented binary files without modification to the original source.

3.2 Analysis

Depending on the configuration settings provided to TAU, it can generate a wide variety of performance data. As shown in fig 1, TAU includes utilities to convert both its profile and trace output to a diverse array of other performance data formats, allowing performance analysis and visualization in many third party performance analysis programs.

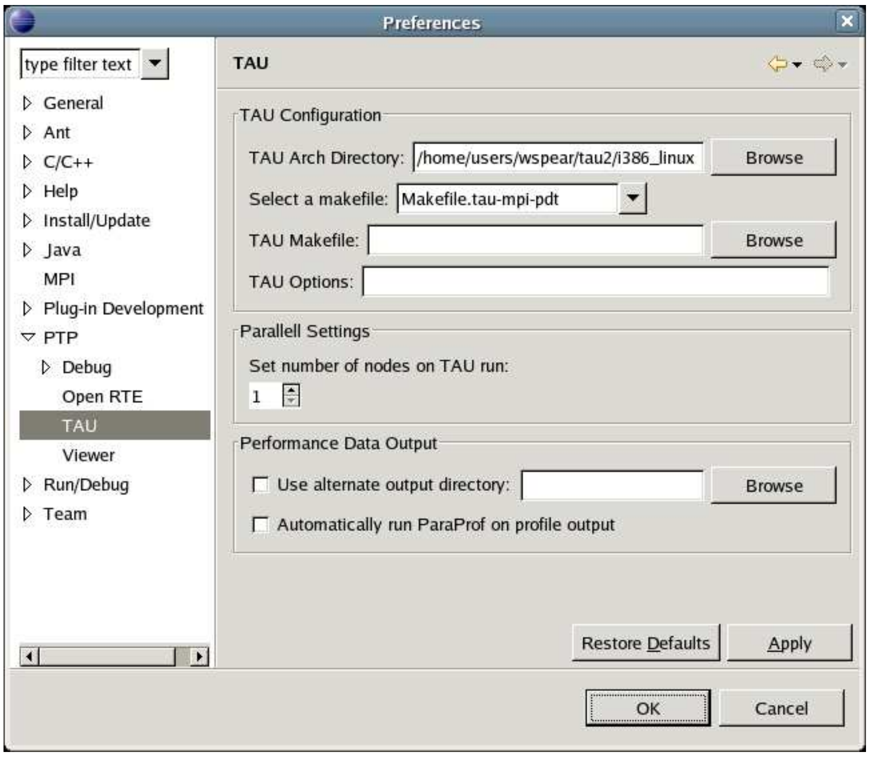
Additionally, TAU includes its own facilities for analysis of performance data. The ParaProf[0] profile analysis tool (seen in Figure 2), for example, provides a full set of graphical tools for evaluation of performance profile data.

4 Integration

Currently, three separate TAU plugins have been developed for Eclipse. Each allows performance analysis within the scope of a different Eclipse IDE implementation, one for the JDT, one for the CDT and one for the PTP. The TAU JDT plugin requires only the standard Eclipse SDK distribution and allows TAU analysis of Eclipse Java projects. The TAU CDT and TAU PTP plugins allow performance evaluation of C and C++ programs within the standard, sequential, C/C++ IDE implementation and the PTP's parallel implementation respectively. Both the TAU CDT and TAU PTP plugins support Fortran when the FDT plugin is installed.

All of the TAU Eclipse plugins share a similar user interface. As shown in Figure 1, a preferences screen allows the user to specify the location of TAU's installation. When this is done a TAU stub makefile can be selected. The makefile will determine which TAU libraries are included in the program's compilation. The preferences also include options for the output location of trace or profile data, the automatic deployment of the profile analysis tool ParaProf on profile output after program execution and other options relevant to the specific language or platform being analyzed.

TAU enabled compilation is invoked in the build system via a context menu in the project explorer, available for objects eligible for standard build operations. Similarly, executable objects are provided with a menu option that allows them to be run with TAU-relevant options automatically. Essentially, once the TAU plugins for the desired IDEs have been installed and configured, obtaining performance data from an Eclipse project is simplified to a sequence of mouse clicks.


Preferences

TAU

TAU Configuration

TAU Arch Directory: /home/users/wspear/tau2/i386_linux

Select a makefile: Makefile.tau-mpi-pdt

TAU Makefile:

TAU Options:

Parallell Settings

Set number of nodes on TAU run: 1

Performance Data Output

☐ Use alternate output directory:

☐ Automatically run ParaProf on profile output

Fig. 1. The preferences screen for the TAU PTP plugin.

4.1 The TAU JDT Plugin

The TAU plugin for the Java Development Tools (JDT) adds an *Instrument* command to the context menu of each Java project, package and source file. This command automatically instruments all of the Java source under the selected object. Thus, in addition to instrumenting an entire project, performance analysis can be done selectively at the source file or package level.

The plugin API of Eclipse's Java Development Tools (JDT) includes access to an Abstract Syntax Tree representation of Java source files. The TAU JDT plugin uses the AST to determine the proper instrumentation points within the source files.

The Java file, package and project context menus are also provided with an uninstrument Java command. This strips all automatically inserted TAU instrumentation from the selected Java file or files.

The TAU JDT plugin supports use of a selective instrumentation file, specified in the TAU JDT options panel. Files and methods can be specified for inclusion or exclusion from instrumentation. The selective instrumentation file supports wild card characters as well. The instrumentor simply accepts or rejects methods and files for instrumentation by referencing the names in the file against those returned by the AST.

Standard instrumentation of Java with JVMPI[0] incurs the overhead of placing probes at each method entry and exit. The TAU JDT's selective source instrumentation avoids this problem. The automatic Java instrumentation management achieved using Eclipse is an example of functionality not available in TAU on its own.

The *Run Tau Instrumented Java* command creates and executes a Java run configuration modified to accommodate the requirements of TAU's invocation, specifically the inclusion of the TAU Java libraries. It builds a directory structure to contain the performance information generated by the instrumented program. It also generates and deploys the command line and environment arguments that specify the location of the required TAU libraries and the output location of the performance data.

4.2 The TAU CDT Plugin

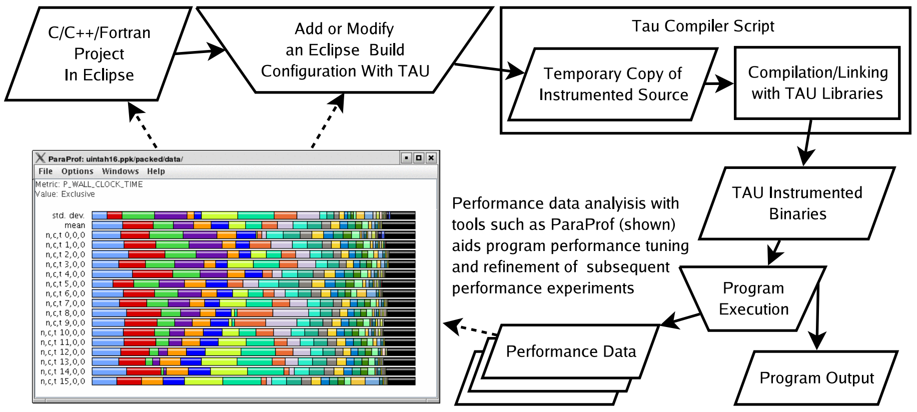

C/C++/Fortran Project In Eclipse
Add or Modify an Eclipse Build Configuration With TAU
Tau Compiler Script
Temporary Copy of Instrumented Source
Compilation/Linking with TAU Libraries
TAU Instrumented Binaries
Program Execution
Program Output
Performance Data
Performance data analysis with tools such as ParaProf (shown) aids program performance tuning and refinement of subsequent performance experiments
ParaProf: uintah16.ppk/packed/data/
File Options Windows Help
Metric: P_WALL_CLOCK_TIME
Value: Exclusive

Fig. 2. TAU CDT/PTP plugin functionality

The C/C++ Development Tools (CDT) project's interaction with TAU differs significantly from the JDT's. TAU's compiler script in conjunction with the TAU stub makefile of an appropriate configuration will automatically perform all of the necessary instrumentation, compilation and linking operations without affecting the source code of the project being analyzed.

The TAU CDT plugin adds a context menu option to Managed Make CDT projects, *Add TAU Configuration*, to create a new build configuration. When this is selected the plugin provides a list of existing build configurations to use as a template. The selected build configuration will be duplicated but with a TAU compiler script specified as the compiler and linker. The CDT plugin interacts with the Fortran Development Tools (FDT) identically to the CDT except that the TAU compiler script replacing the default compiler is Fortran specific. The compiler script is passed the stub makefile chosen in the TAU preferences screen. Because the compiler script recognizes compilation and linking commands as if it were a conventional compiler any preexisting customizations to the selected build configuration will be properly included in the new TAU build configuration. The compiler script's options, which may be set at the plugin preferences screen, include selective instrumentation similar to that described for the JDT.

The *Run Tau* command added to the binary context menu operates analogously to that provided by the JDT plugin. However, because the TAU libraries are included at compilation, the standard run system can be used without providing any TAU specific options. Figure 2 outlines the steps in obtaining performance data from an eclipse project via the plugin.

4.3 The TAU PTP Plugin

For the most part, the Parallel Tools Project (PTP) build system is equivalent to the CDT. Internally, the interfaces with the PTP build and run systems are similar to those of the CDT. The primary distinction is the necessity of setting a default number of processes with which to run the parallel code, as specified in the plugin's options.

As the PTP project develops, further enhancements of the TAU PTP plugin may be required. For example when the PTP implements remote execution of parallel computing jobs the TAU plugin will require a mechanism of obtaining the resulting performance data from the remote system.

Because program performance is an essential goal of high performance computing the TAU PTP plugin is likely to be the main focus of future development. Fortunately, its architectural similarity to the TAU CDT plugin may eventually allow the two to be combined, simplifying deployment and maintenance in the future.

4.4 Future Work

The present implementation of the TAU plugins allows access to most of TAU's capabilities from within Eclipse. However there are several avenues of integration and interface development that remain to be explored. Additionally there may be useful functions that can be developed within the Eclipse framework that have no parallel within the standalone TAU system.

One of the most immediate goals of future TAU plugin development is addition of persistent source instrumentation for projects within the CDT, FDT or PTP, similar to that already implemented in the JDT plugin. TAU's compiler script does offer some control over project instrumentation but allowing persistent instrumentation in the source code would greatly facilitate manual adjustment of that instrumentation. Manual instrumentation allows a finer level of user control which may be desirable in some performance analysis scenarios. Persistent source instrumentation would also necessitate some functionality to manage and allow switching between instrumented and uninstrumented code.

Another goal is to move the integration of TAU and Eclipse to a lower, more internalized level. Ideally, instead of calling on TAU's compiler scripts, the CDT build system will derive that functionality from the TAU plugin itself. Only the TAU makefiles and libraries would be required to build and run a TAU instrumented project once this is accomplished.

An important feature yet to be implemented is advanced management of performance data output. Ultimately the directory based solution will be replaced by a performance database solution within Eclipse analogous to the PerfDMF[0] system already included with the TAU.

In general, closer unification of TAU's peripheral utilities with Eclipse is a priority. For example allowing ParaProf to communicate with Eclipse could permit performance hot-spots depicted in ParaProf to programmatically link back to their source calls in Eclipse.

5 Conclusion

Growing interest in the use of IDEs for parallel computing raises questions regarding how parallel tools will be integrated. The nature of these tools, the variations specified in their software development and platform implementation, and the complexity of their function suggests that re-engineering of the tools to make them more compatible with the IDE framework would be a major undertaking. Instead, an approach that looks at the problem from a point of view of tool interoperability is more productive. We have described in this paper our approach for the integration of TAU with Eclipse following this strategy.

By providing a clean, simple interface between TAU and the Eclipse JDT, CDT and PTP projects we have created a relatively straightforward, unified mechanism of undertaking performance analysis within these respective Eclipse IDEs. High performance computing applications in particular require performance analysis and tuning during their development life-cycle to to achieve their implicit goal of efficient operation. Of course, conventional, sequential programs in more common venues also stand to benefit from IDE-aided performance analysis and optimization. In either case a consistent and robust mechanism for performance analysis is advantageous during the software development process.

Because TAU and Eclipse are only loosely coupled, significant changes in the functionality or interface of either of these actively developing programs is unlikely to disrupt of their integration. Maintenance and continued enhancement of the TAU plugins as described above will continue. TAU Eclipse integration will continue supporting IDE assisted performance analysis within Eclipse as the Eclipse platform and its IDE components continue to mature and as TAU's capabilities extend to new parallel analysis paradigms.

References

1. Amsden, J., "Levels Of Integration: Five ways you can integrate with the Eclipse Platform", http://www.eclipse.org/articles/Article-Levels-Of-Integration/levels-of-integration.html, March 2001

2. R. Bell, A. D. Malony, and S. Shende, "A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis", Proc. EUROPAR 2003 conference, LNCS 2790, Springer, Berlin, pp. 17-26, 2003
3. CDT - C/C++ Development Tools, http://www.eclipse.org/cdt
4. Eclipse, http://www.eclipse.org
5. K. Huck, A. Malony, R. Bell, L. Li, and A. Morris. PerfDMF: Design and implementation of a parallel performance data management framework. In Proc. International Conference on Parallel Processing (ICPP 2005). IEEE Computer Society, 2005
6. JDT - Java Development Tools, http://www.eclipse.org/jdt
7. K. A. Lindlan, J. Cuny, A. D. Malony, S. Shende, B. Mohr, R. Rivenburgh, C. Rasmussen. "A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates." Proceedings of SC2000: High Performance Networking and Computing Conference, Dallas, November 2000.
8. Open MPI, http://www.open-mpi.org
9. Popescu, V. "Java Application Profiling using TPTP.", Eclipse Corner Article, http://www.eclipse.org/articles/Article-TPTP-Profiling-Tool/tptpProfilingArticle.html, Febuary 2006
10. PTP - Parallel Tools Platform, http://www.eclipse.org/ptp
11. S. Shende and A. D. Malony, "The TAU Parallel Performance System," International Journal of High Performance Computing Applications, ACTS Collection Special Issue, 2005
12. SUN Microsystems Inc., Java Virtual Machine Profiler Interface (JVMPI), http://java.sun.com/j2se/1.3/docs/guide/jvmpi/jvmpi.html
13. TAU - Tuning and Analysis Utilities, http://www.cs.uoregon.edu/research/tau/home.php
14. TPTP - Test and Performance Tools Platform, http://www.eclipse.org/tptp