Parametric Studies in Eclipse with TAU and PerfExplorer

Kevin A. Huck, Wyatt Spear, Allen D. Malony,
Sameer Shende and Alan Morris

Performance Research Laboratory
Department of Computer and Information Science
University of Oregon, Eugene, OR, USA
{khuck,spear,malony,shende,amorris}@cs.uoregon.edu
http://www.cs.uoregon.edu/research/tau

Abstract. With support for C/C++, Fortran, MPI, OpenMP, and performance tools, the Eclipse integrated development environment (IDE) is a serious contender as a programming environment for parallel applications. There is interest in adding capabilities in Eclipse for conducting workflows where an application is executed under different scenarios and its outputs are processed. For instance, parametric studies are a requirement in many benchmarking and performance tuning efforts, yet there was no experiment management support available for the Eclipse IDE. In this paper, we describe an extension of the Parallel Tools Platform (PTP) plugin for the Eclipse IDE. The extension provides a graphical user interface for selecting experiment parameters, launches build and run jobs, manages the performance data, and launches an analysis application to process the data. We describe our implementation, and discuss three experiment examples which demonstrate the experiment management support.

Keywords: parallel, performance, experiment management, Eclipse.

1 Introduction

Integrated development environments (IDEs) help to facilitate software development and maintenance. IDEs provide a consistent development environment, numerous enhancements to the development process, and are the standard in industrial software development. IDEs are not very common in parallel application development, but improving toolkit functionality makes it possible to write, compile, launch and run large-scale parallel applications on a local machine or on a remote resource.

Parametric studies are necessary in many benchmarking and performance tuning efforts. This is especially true for parallel applications, where scaling studies are key to exploiting highly parallel hardware for maximum return. Parametric studies are helpful in finding scaling bottlenecks and communication design flaws, and improving algorithmic efficiency. However, parametric studies can

consist of hundreds or thousands of application configurations, and automated parametric studies can be complex to perform. Traditional parmetric studies on parallel hardware requires scripts for building configurations, scripts for submitting batch jobs to the queue, scripts for data management, and the eventual analysis processing at the end of the executions. Script programming is error prone, and particularly costly if mistakes are not found until after hundreds of processing hours have been consumed. Parametric study scripts are frequently re-usable only with considerable effort, as the differences between two or more parallel applications can be significant. There is a clear opportunity to improve upon the parametric study process.

Eclipse [1] is a user configurable software development IDE with a plugin-centric design. Plugins have been developed for a wide range of development purposes. A TAU [2] performance analysis toolkit plugin for Eclipse has been written, and can be used for instrumentation and measurement of C, C++, Fortran and Java applications developed in Eclipse [3, 4]. However, there was no mechanism in the various plugins for experiment management with regards to performance studies. For that reason, we extended the Parallel Tools Platform (PTP) plugin for Eclipse to include a parametric study framework for the TAU plugin in Eclipse. Users can set up a desired study, launch the experiment, and the framework will automatically compile and execute the application with the specified configuration combinations, storing the performance result after each run in a PerfDMF [5] repository. When the experiment is complete, the multi-experiment analysis and data mining tool PerfExplorer [6] is launched, and the automated comparative analysis results are produced. While TAU, PerfDMF and PerfExplorer are the tools we used in our experiments, consideration was given to the implementation to ensure that other performance tools could be used in the same framework.

The remainder of this paper is as follows. Section 2 will provide some background discussion of Eclipse and the plugin components involved. Section 3 will describe the experiment management support implementation. Section 4 will describe some analysis examples using the experiment management support. Section 5 will describe related work in parametric study support, and Section 6 will describe our conclusions and future work.

2 Background

Eclipse Eclipse[1] 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 has provided support for other languages such as C/C++ and Fortran. Support for high performance computing has also been provided via the Parallel Tools Platform (PTP). Two distinct advantages of the Eclipse platform are its portability and extensibility. The former is provided largely by Eclipse's Java-based implementation, which means it can be run consistently on Windows, Macintosh and many Unix based OSes. Because Eclipse is open source, users are free to modify and extend its functionality as they see fit. As a

result, there is a diverse array of enhancements and plugins available to increase Eclipse's functionality for software development, as well as other tasks. A longer description of Eclipse can be found in [3, 4] and elsewhere.

Eclipse Plugins There are four Eclipse plugin collections, or projects, that are related directly to the integration of the TAU performance analysis tools. The Java Development Tools (JDT) [7], the C/C++ Development Tools (CDT) [8], the Photran Development Environment [9], and the Parallel Tools Platform (PTP) [10] 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 assists with Java development by providing a context sensitive source editor, project management and development control facilities, among other features.

CDT Many of the CDT's features are comparable to those of the JDT. 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.

Photran Photran is a Fortran development environment, based on CDT. Photran has support for Fortran 77, 90 and 95.

PTP PTP provides the ability to write, compile and launch and debug parallel programs from within Eclipse. PTP supports both OpenMP and MPI based parallelism, and is also based on CDT.

TAU TAU [2] 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.

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. TAU includes utilities to perform automatic instrumentation of source code. TAU 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.

Analysis Depending on the configuration settings provided to TAU, it can generate a wide variety of performance data. 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. Performance profiles are automatically uploaded to a PerfDMF [5] repository for analysis.

Additionally, TAU includes its own facilities for analysis of performance data. The ParaProf[11] profile analysis tool, for example, provides a full set of graphical tools for evaluation of performance profile data. PerfExplorer [12] is a multi-experiment analysis and data mining tool, designed to provide parametric study analysis and intelligent analysis of results using performance data, metadata, analysis scripts, and inference rules.

TAU Plugin for Eclipse PTP 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 Photran plugin is installed.

The TAU CDT and PTP plugins extend the CDT and PTP launch configuration systems, respectively. They allow the selection of a TAU stub makefile, which will determine which TAU libraries are used at the program's compilation. The plugins also allow specification of selective instrumentation, other TAU-specific compilation options and data collection options.

When an application is launched within Eclipse using the TAU plugins an instrumented executable will be generated and run using the selected options. This executable will then be run by the CDT or PTP launch management system. When execution is complete profile data may be stored automatically in a local PerfDMF database and viewed in ParaProf. 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. A complete description of the plugins can be found in [3].

3 Design and Implementaion

The initial implementation of the performance analysis work-flow system followed a linear three-step process. Compilation, execution and analysis were per-

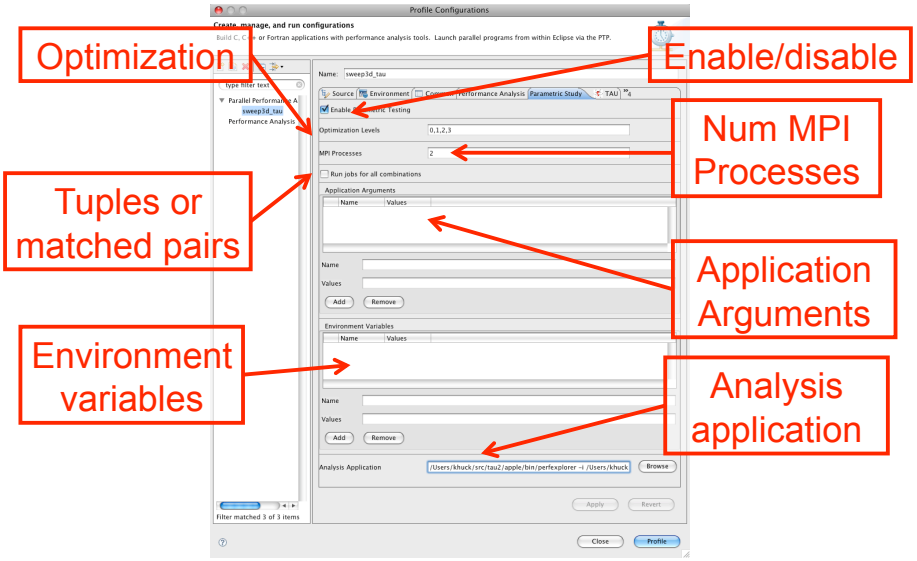
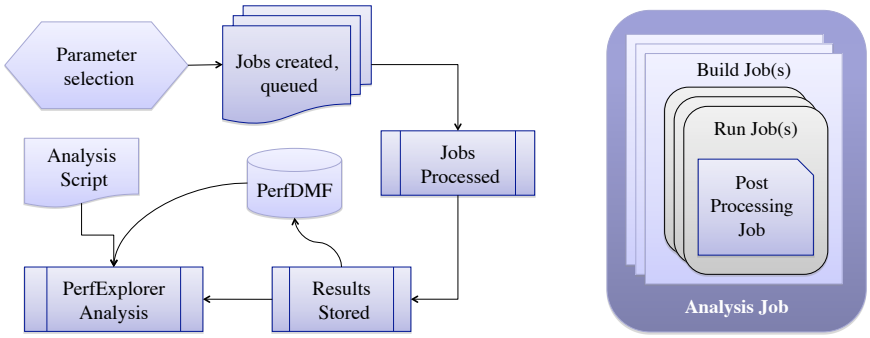


Fig. 1. The Experiment Management user interface.

formed using the parameters specified by the user in the launch configuration interface. Each step was handled by the job management system provided by the Eclipse API. However the respective jobs made assumptions about their execution order. To add support for multiple parametric combinations it was necessary to make significant changes to the existing work-flow system.

The first such change was to allow the user to specify lists of parameters in a new UI component specific to parametric analysis, as showin in Figure 1. The values available for parametrization are build optimization options, processor count, application arguments and run-time environment variables. Initially we generated one set of parameters for each combination of list elements. Subsequently we added the ability to limit parameter sets to those required such as for weak scaling studies. This was accomplished by creating tuples from the parameter lists, where each parameter list had the same number of potential values. Each index of the parameter lists were matched up (i.e. the first parameter value in each populated list for one experiment, the second parameter value in each populated list for the second experiment, etc.) In the case of weak scaling studies, for each of M processor count values, there would be N input problems (where $M = N$) so rather than generating $M * N$ experiments, there are only N experiments.

The procedure for each parameter set generated is similar to the linear system used in the earlier implementation. However, some modifications were needed to improve efficiency and provide each step with data required from previous steps. For example, because the build and launch steps are independent in the Eclipse environment, only one build operation needs to be performed for each



(a) Parametric study workflow.

(b) Nested, queued jobs.

Fig. 2. 2(a) shows the overall parametric study workflow. 2(a) shows the nesting of jobs. There is an outer loop of build jobs and an inner loop of run jobs. Each run is followed by a post-processing step, where performance data is uploaded to PerfDMF. After all build and run jobs are complete, the analysis job is run.

set of build parameters. Each combination of launch parameters is run once on a single executable generated by a build step. Then the executable is rebuilt with the next set of build parameters and the process repeats.

The single data structure previously responsible for management of the build, execution, and analysis steps was divided into three task-specific components to better support the decoupling of the steps in parametric analysis. Because the launch and analysis jobs relied on data created by the build job, it had been necessary to perform a full build before performing any of the subsequent steps. Thus, another change necessitated by the more flexible parametric analysis system was to initialize the relevant data in the build job object upon its instantiation rather than upon being run. This enables each job to be created and put into a queue before any job is run. The jobs can then be run in succession. A similar dependency existed between the launch job and the post-launch job - the performance data cannot be archived until the execution is complete. These architectural changes will help with the future addition of more complex work-flow capabilities.

The full operating procedure of the parametric analysis system, shown in Figure 2, is as follows. Given the lists parameters from the user, a list of distinct build and launch parameter sets is generated. For each build parameter set in the list, a build job is created and placed in a job queue. After every such build job, for each launch parameter set, launch and analysis jobs are created and placed in the queue. When the queue is fully populated the first job is run. The last action of each job is to initiate the subsequent job in the queue. The final

analysis job in the queue contains additional instructions to launch any final analysis operations on the whole of the data generated by the parametric run.

As of this writing the parametric analysis system's support for user specification of build parameters remains rudimentary. The build system provided by Eclipse contains compiler-specific options which must have valid values selected. Thus, providing a relatively simple UI for specification of arbitrary build parameters requires some foreknowledge of the implementation of Eclipse build-chains for specific compilers. This is in contrast to the launch configuration system where the user may specify arbitrary strings as environment variables or program arguments.

The parametric analysis system was developed to inter-operate directly with the TAU performance analysis system. However the Eclipse performance framework is designed for more general applications. Presently the TAU plug-in is the only component of the performance framework which fully exploits the capabilities of the parametric analysis system. Fully incorporating support for arbitrary analysis tools remains a priority.

4 Examples

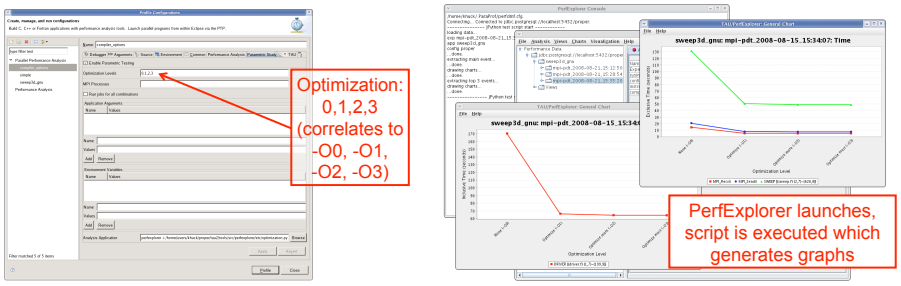
To demonstrate the functionality of the experiment management support we added to Eclipse PTP, we constructed a number of parametric study examples. In this section, we discribe two applications, Sweep3D and LU from NPB3.2.1, used in three different parametric studies.

4.1 Sweep3d

Sweep3D [13] solves a 1-group, time-independent, discrete ordinates, 3D Cartesian geometry neutron transport problem. The main algorithm is a wavefront process across the I and J dimensions, and is pipelined along the K dimension. The algorithm gets its parallelism from the I, J domain decomposition. Sweep3D is written in Fortran 77, and uses MPI. There is also a timer routine written in C, but In this experiment, the timer routine was disabled, as we were using TAU for instrumentation. The code was also modified to take the name of the input file as a command line argument, to allow for parametric studies. This was necessary, as the input file also specifies the domain decomposition in each direction, and the total number of processes has to match the number of MPI processes.

The first parametric study was a compiler optimization study. Sweep3D was compiled with the GNU Fortran compiler [14], using four different optimization settings: -O0 (no optimization), -O1 (some optimizations), -O2 (more optimizations), and -O3 (most optimized). The results of the parametric study are shown in Figure 3.

In order to process the experiment, four build jobs were automatically constructed, each with a different compiler optimization setting. Each of the build jobs had a corresponding run job, and a post-run job. The post-run job for each



(a) Compiler Options selected in Eclipse.

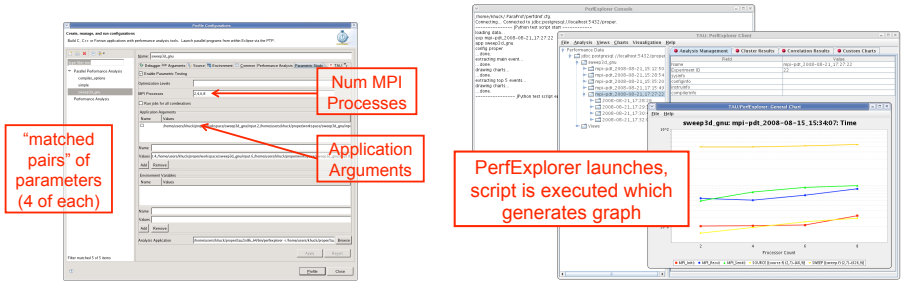
(b) PerfExplorer output.

Fig. 3. The results of running the compiler optimziation study - PerfExplorer is executed, and the results of the study are visualized in two charts.

execution was to save the TAU performance profiles into the PerfDMF database for that application, creating a new experiment in the database. After all of the build and run jobs were complete, there was one final post-processing job which ran PerfExplorer and executed an analysis script. The script loaded the performance data into PerfExplorer and generated two charts, one showing the total runtime for the application for the four optimization settings, and one chart showing the runtime for the three most time-consuming instrumented regions in the application.

The second parametric study was a weak scaling study. This was an interesting challenge, as for each "number of processors" value, there was a corresponding input problem which scaled at the same rate as the number of MPI processes. In our study, we used 100 grid cells per processor in each direction, and 100 planes in the K direction. Therefore, for 2, 4, 6, and 8 processors, the problem size was 20,000, 40,000, 60,000 and 80,000 total grid cells, respectively. The results of the study are shown in Figure 4.

In order to process the experiment, one build job was constructed, as only one was necessary. Four run jobs were created, each with a different number of MPI processes and a corresponding input file. Again, each of the run jobs had a corresponding post-run job to save the performance profiles to PerfDMF. After the one build and all four run jobs were complete, there was one final post-processing job which ran PerfExplorer and executed an analysis script. The script loaded the performance data and generated one chart showing the runtime for the five most significant instrumented regions in the application. The chart demonstrated that there was a slight increase in overhead in the main computation routines (not uncommon, but ideally the time spent in computation would stay level for all numbers of processors), and an increase in overhead in the MPI communication routines, which is expected in scaling studies.



(a) MPI weak scaling options in Eclipse.

(b) PerfExplorer output.

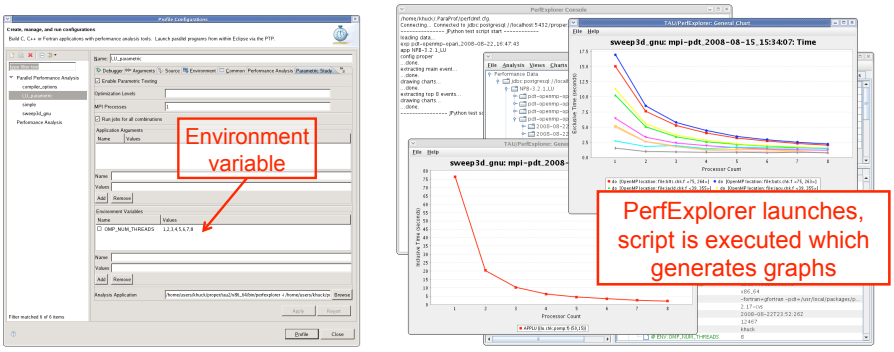
Fig. 4. The results of running the MPI weak scaling study - PerfExplorer is executed, and the results of the study are visualized in one chart.

4.2 NPB 3.2.1 LU

The NAS (NASA Advanced Supercomputing) Parallel Benchmarks (NPB)[15] are a set of programs designed to evaluate the performance of parallel systems. NPB 3.2.1 includes serial, MPI and OpenMP implementations of computational fluid dynamics algorithms. We elected to use the LU benchmark in order to evaluate the OpenMP parallelism functionality in the experiment management system. The LU benchmark uses successive over-relaxation to solve a diagonal system by splitting it into block lower and upper triangular systems.

Our third parametric study was a strong scaling study. In this study, we varied the number of threads available to the OpenMP runtime, and compared the results from each execution of the benchmark. In our study, we constructed the "A" class problem, which solves a $64 * 64 * 64$ system in 250 iterations. We requested a study with all integer values of threads between one and eight, inclusive. The results of the study are shown in Figure 4.

In order to process the experiment, only one build job was constructed. Eight run jobs were created, each with a different number of OpenMP threads. This was accomplished by setting the `OMP_NUM_THREADS` variable to a different value for each run. As with the other examples, each of the run jobs had a corresponding post-run job to save the performance profiles to PerfDMF. After the one build and all eight run jobs were complete, there was one final post-processing job which ran PerfExplorer and executed an analysis script. The script loaded the performance data and generated two charts, one showing the total runtime for the application for each of the eight thread values, and one chart showing the runtime for the ten most time-consuming instrumented regions in the application.



(a) OpenMP strong scaling options in Eclipse.

(b) PerfExplorer output.

Fig. 5. The results of running the OpenMP strong scaling study - PerfExplorer is executed, and the results of the study are visualized in two charts.

5 Related Work

There are a few related experiment management systems, but to the best of our knowledge, none of them are integrated into an application development environment such as the Eclipse IDE, nor do they support resource managers on large shared systems.

Prophesy [16] is an online database which uploads and stores data from instrumented parallel application runs. Prophesy applies the performance database to manage multi-dimensional performance information for parallel analysis and modeling. The data is accessible from a web interface, and various models can be built, including curve fitting, parametric models, and kernel coupling. There are data generation and submission (of the performance data to the Prophesy database) utilities as part of Prophesy, but they are not automated.

Pythia-II [17] is a system for generating performance data from a large parametric space, with the goal of recommending optimized solutions to developers from a number of alternatives. The Pythia-II system combines knowledge discovery with recommender systems to mine performance data, and provide runtime selection of application parameters. While Pythia-II has software for generating performance data, it is for the purpose of searching within the recommender system, and is not intended for general purpose parametric studies.

ZENTURIO [18] is an environment for generating parametric studies for parallel performance analysis. On completion of an application execution, the performance data is automatically stored in a repository. The environment includes a performance visualizer which can perform multi-experiment analysis. ZENTURIO includes support for cluster and Grid computing.

Aksum [19] is a related multi-experiment performance analysis tool which automatically instruments an application, builds and executes the application with given parametric values, and analyzes the results with the goal of locating performance bottlenecks.

6 Conclusion

The work reported here demonstrates initial steps toward integrating automated performance analysis in the Eclipse IDE. In particular, we showed how parametric studies are made easier with Eclipse PTP and Experiment Management support. Support for optimization settings, MPI processor counts, environment variables, and application arguments were developed in Eclipse and used to generate experiments for execution with the TAU Performance System. The project also expanded and improved the automation of analysis results using PerfExplorer scripts.

While successful, there are a few open issues and related problems for our chosen solution. The Eclipse PTP support for launch managers is limited, and not yet robust unless specific versions of supported MPI libraries are used. There are four resource managers supported in Eclipse PTP (ORTE [20], IBMLL [21], PE [22], MPICH2 [23]), but they were either not in use on our parallel systems, or did not work as advertised. For this reason, despite the fact that we had a 128-core system at our disposal, we were unable to perform parametric studies which ran on more than one node. This problem will be resolved with more robust support in the future.

We are also investigating better ways to handle unusual combinations of parameters. Currently, either all combinations of all parameter values or matched pairs are supported. Other possible ways to specify parameters include value ranges or algorithmic expressions. There are also questions about whether values in the experiment management suppport override the values in the application build and run configurations, or add to them. Flexible parameter mechanisms would allows the tools to be more broadly applied. In addition, the larger problem of specifying complex combinations of parameters to the build process is as yet unresolved, as described in Section 3.

In conclusion, we belive that as the Eclipse PTP support continues to improve, we will see more parallel application development in Eclipse, and parametric studies submitted from IDEs will become more commonplace.

7 Acknowledgments

University of Oregon research is sponsored by contracts DE-FG02-07ER25826 and DE-FG02-05ER25680 from the MICS program of the U.S. DOE, Office of Science and NSF grant #CCF0444475.

References

1. The Eclipse Foundation: Eclipse.org Home. `http://www.eclipse.org` (2008)
2. Shende, S., Malony, A.D.: The TAU parallel performance system. The International Journal of High Performance Computing Applications **20**(2) (Summer 2006) 287–331
3. Spear, W., Malony, A.D., Morris, A., Shende, S.: Integrating TAU with Eclipse: A Performance Analysis System in a Integrated Development Environment. In: High Performance Computing and Communications (HPCC) Conference. Volume 4208/2006 of LNCS., Springer (September 2006) 230–239
4. Spear, W., Malony, A.D., Morris, A., Shende, S.: Performance Tool Workflows. In: International Conference on Computational Science (ICCS) *in publication*. (June 2008)
5. Huck, K., Malony, A., Bell, R., Morris, A.: Design and Implementation of a Parallel Performance Data Management Framework. In: Proceedings of the International Conference on Parallel Computing, 2005 (ICPP2005). (2005) 473–482
6. Huck, K.A., Malony, A.D., Shende, S., Morris, A.: Scalable, Automated Performance Analysis with TAU and PerfExplorer. In: Parallel Computing (ParCo), Aachen, Germany (2007)
7. The Eclipse Foundation: Eclipse Java Development Tools (JDT) Subproject. `http://www.eclipse.org/jdt` (2008)
8. The Eclipse Foundation: Eclipse C/C++ Development Tooling - CDT. `http://www.eclipse.org/cdt` (2008)
9. The Eclipse Foundation: Photran - An Integrated Development Environment for Fortran. `http://www.eclipse.org/photran` (2008)
10. The Eclipse Foundation: PTP - Eclipse Parallel Tools Platform. `http://www.eclipse.org/ptp` (2008)
11. Bell, R., Malony, A., Shende, S.: A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis. In: Proc. EUROPAR 2003 Conference (EUROPAR03). (2003)
12. Huck, K.A., Malony, A.D., Shende, S., Morris, A.: Knowledge Support and Automation for Performance Analysis with PerfExplorer 2.0. Scientific Programming, special issue on Large-Scale Programming Tools and Environments **16**(2-3) (2008) 123–134
13. LLNL: The ASCI Sweep3D Benchmark. `http://www.llnl.gov/asci/purple/benchmarks/limited/sweep3d/` (2006)
14. Free Software Foundation, Inc.: GNU Fortran - Free Number Crunching FOR All! `http://www.gnu.org/software/gcc/fortran/` (2008)
15. Bailey, D., Harris, T., Saphir, W., van der Wijngaart, R., Woo, A., Yarrow, M.: The NAS Parallel Benchmarks 2.0. Technical Report Technical Report NAS-95-020, NASA Ames Research Center (December 1995)
16. Taylor, V., Wu, X., Stevens, R.: Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications. SIGMETRICS Perform. Eval. Rev. **30**(4) (2003) 13–18
17. Houstis, E.N., Catlin, A.C., Rice, J.R., Verykios, V.S., Ramakrishnan, N., Houstis, C.E.: PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software. ACM Trans. Math. Softw. **26**(2) (2000) 227–253
18. Prodan, R., Fahringer, T.: On Using ZENTURIO for Performance and Parameter Studies on Cluster and Grid Architectures. In: 11th EuroMicro conference on Parallel Distributed and Network-Based Processing (PDP 2003). (February 2003)

19. Fahringer, T., Clovis Seragiotto, J.: Aksum: a performance analysis tool for parallel and distributed applications. (2004) 189–208
20. The OpenMPI Project: Open MPI: Open Source High Performance Computing. http://www.open-mpi.org/ (2008)
21. IBM: IBM Redbooks — Workload Management with Load Leveler. http://www.redbooks.ibm.com/abstracts/SG246038.html (2008)
22. IBM: IBM POWER processor-based servers: Software, Parallel Environment (PE). http://www-03.ibm.com/systems/p/software/pe/index.html (2008)
23. http://www.mcs.anl.gov/research/projects/mpich2/: MPICH2: High-performance and Widely Portable MPI. http://www.mcs.anl.gov/research/projects/mpich2/ (2008)