A Strategy for Addressing the Needs of Advanced Scientific Computing Using Eclipse as a Parallel Tools Platform

Gregory R. Watson
Craig E. Rasmussen

Los Alamos National Laboratory
P.O. Box 1663, MS B287
Los Alamos, NM 87545

December 2005

LA-UR-05-9114

ABSTRACT

If parallel computer systems are to achieve the kinds of productivity improvements necessary to meet the needs of high productivity computing systems (HPCS), then a radical change will be required in how tools are developed for programmers and users of these systems. The number and complexity of tools is making it difficult for developers to access and use the tools effectively, and the lack of a common tool infrastructure significantly complicates tool development efforts. The rapid pace of change means that developing and maintaining the kinds of tools that will be needed to effectively utilize the capacity of future advanced computer systems is an increasingly onerous task.

This paper proposes a strategy that will lead directly to productivity and quality improvements in the development and use of parallel applications, and that will provide a framework to guide and foster future tool development. This strategy is based on using the Eclipse platform to form the foundation of an integrated environment for parallel application development. The intention is not just to produce another set of tools, however, but rather to use Eclipse as both a focal point for tool development, and as a platform for producing highly integrated tools that are needed to meet the future needs of the HPC community.

1 INTRODUCTION

As high productivity computing systems (HPCS) are developed to meet the demands and computational challenges facing advanced scientific research, it is becoming increasingly apparent that existing software infrastructure and tools will need to be substantially improved in order to achieve the goal of sustained performance on terascale machines [17].

There are significant limitations with the current development process for high performance applications. As reported in a recent survey of existing tools [5], there is a perception among application developers of a lack of tools supporting high performance computing. However, the truth is that there exist an enormous number of tools, but that developers either do not know about the tools or do not have access to them. The tools themselves are often only available on a small number of platforms, requiring the developer to learn new tools on each new platform. The tools also share virtually no common infrastructure or user interfaces. This significantly complicates tool use and prevents the tools from being easily or effectively incorporated into the application development process. This situation will only be compounded as systems become increasingly powerful and complex, necessitating new and more complex tools. The result will be that it will become more and more difficult to realize the full potential of these machines, leading to wasted resources and a failure to meet the desired productivity improvements.

Rather than developing, yet again, more tools to add to developer confusion, we propose a software infrastructure for tool integration. This infrastructure would provide a common user interface across computing platforms, while remaining agnostic to the actual back-end tools deployed. For example, while compilers, linkers, job schedulers, debuggers, run-time systems, and performance analysis tools are likely to change across platforms, the user interface to these tools should remain largely the same. Further, because the infrastructure would ensure tools could be tightly integrated together, it would directly lead to significant benefits resulting from the sharing of data and functionality between tools. Not only would this reduce the burden on tool developers, but it would also provide unique opportunities for new tool development.

This strategy for tool integration focuses on an integrated development environment (IDE) that encompasses five major areas. In each area we propose a range of tools that could be integrated; these examples are for illustrative purposes only and do not form an exhaustive list. The integration areas are:

Program Development – An environment to support the creation (editing) and building of parallel applications. Tools available should include: language-aware editors for program creation; language parsers supporting program restructuring and refactoring; and an infrastructure supporting static analysis and source-to-source transformations for program parallelization and optimization. Examples of existing tools that could form the basis for tool integration are the Eclipse C/C++ Development Tools (CDT) and Photran

projects[1]. Specific tools and languages that could be integrated to work with the IDE are the Co-Array Fortran [11] and UPC [6] (for parallelization) and the ROSE compiler technologies [10] (for refactoring and optimization).

Launch and Runtime Support – A common runtime architecture that simplifies the interaction with parallel machines and supplies common services such as application launch (e.g. mpirun[2]), interaction with queuing systems (e.g. LSF [13]), the display of job and machine status (e.g. supermon [15]/Ganglia [7]), and runtime support for parallel debugging.

Debugging and Testing – An integrated scalable parallel debugger that employs a common user interface to native, back-end debuggers such as IDB, dbx, gdb, etc. In addition, a common interface to automated unit testing, including test harness generation, test stub generation and test execution and reporting is needed. This unit test infrastructure could be modeled after the Eclipse JUnit project, but utilize the static analysis infrastructure described above for C, C++, and Fortran programs or connect to commercial tools such as Cleanscape's Fortran Grayboxx [3].

Performance Monitoring – A common user interface to performance monitoring tools and associated runtime information and generated files. Existing profiling toolkits that could be integrated include HPC Toolkit [9], TAU [16] and Open SpeedShop [9].

Deployment – A common user interface to support the final deployment of applications and tools. Because the IDE has intimate knowledge of application files and their internal structure (for example, the names of included files and system calls that are invoked), tools such as autoconf, automake, and libtool can be integrated with the development environment and with the build system. The goal would be for the IDE to transparently create the files necessary to deploy an application, in such a way that a application user could build and install the application using the IDE, given only an URL to the application package.

Application developers will see the immediate productivity benefits of using such an integrated environment. These benefits include:

- A simple and consistent way to download, build, and install required software tools.
- A consistent and customizable interface for developing, launching and debugging programs as well as monitoring job status.
- Simplified application portability across parallel machines and execution environments.

[1] The authors have contributed to both projects. See http://eclipse.org/cdt and http://eclipse.org/photran
[2] A standard command provided by most MPI implementations

- Advanced features that simplify the development process, such as language-aware editors and other views to show and manipulate program structure, automatic code generation (e.g., for unit testing), and an integrated and automated build system.

Tool developers will gain the same advantages as application developers, but in addition will benefit from a range of other features, including:

- Assistance for tool packaging and deployment so that tool users can automatically download, build and install tools given the URL to a tool package.

- The ability to utilize common and portable services for tool development, rather than having to develop such services from scratch.

- Access to a portable, highly customizable graphical user interface, and the ability to reuse and share user interface elements.

- Functionality that allows data sharing between tools, and permits tools to utilize each other's services.

In the remainder of this paper, we will outline the proposed approach in more detail, describe how existing tools can be integrated into the architecture, and provide some concrete examples of integrating specific tools.

2 THE ECLIPSE PARALLEL TOOLS PLATFORM

Eclipse is a Java-based platform that was designed to enable the integration of a wide range of programming and application development tools into a portable, scalable and open source integrated development environment. In 2001, IBM donated the Eclipse platform to the Eclipse Foundation, a non-profit corporation formed to advance the creation, evolution, promotion, and support of the Eclipse platform and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services[1]. The Foundation has a membership that now exceeds over 100 organizations, and each make substantial contributions in the form of both ongoing revenue and developer resources. The Eclipse Foundation has established policies in relation to intellectual property and copyright[2], corporate governance, and a developer meritocracy. A wide variety of commercial products are now based on the Eclipse business model, which allows both open-source and closed-source components to be integrated into the Eclipse platform.

The Eclipse Parallel Tools Platform (PTP) project was established in January 2005 with the goal of producing an open-source industrial-strength platform that provides a highly integrated environment specifically designed for parallel application development. The architecture of the Parallel Tools Platform consists of three main components:

[1] For more information see http://www.eclipse.org/org/
[2] Eclipse source code is released under the open-source Eclipse Public License (EPL)

- **Parallel User interface** – Extensions to the Eclipse IDE to allow the interaction with arbitrary parallel systems for the launching, control, monitoring and debugging of parallel applications.

- **Parallel Debugger** – An integrated parallel debugger that allows the debugging of large parallel and message passing codes.

- **Parallel Runtime** – A standard interface to external parallel runtime systems that allows applications and tools to interact with a variety of parallel machines using a single common mechanism.

By integrating with Eclipse, the Parallel Tools Platform automatically inherits a range of services that are essential to improving the productivity of application development. These include:

- **Build system** – Eclipse provides the ability to utilize existing (make-based) build systems, as well as providing a managed build system that can be used to automate the build process.

- **Language aware editors** – Eclipse includes compiler front-ends for a range of languages, including C, C++, Fortran and Java. These are combined with editors to provide a range of language services, such as syntax coloring, on-the-fly syntax checking, code assist, language-specific searching, outline view, and code refactoring.

- **Multiple language support** – Native parsers and related support for C, C++, Fortran and Java is provided, and there are extensions to support scripting languages such as Python, Perl and Ruby.

- **Source code control** – Integration with CVS is a standard feature, and there are extensions to support other source code control systems, such as Subversion.

- **Integration with external development tools** – Eclipse provides interfaces to standard compilers and debuggers so that they can be accessed transparently through the user interface and build system.

PTP is an ongoing project that has strong backing from a number of the DARPA HPCS Phase II Industry Teams[1], and is beginning to see contributions from some other major vendors. The design and development of PTP is also closely aligned with the objectives of the tool integration strategy, which makes it the ideal choice as a parallel tool integration platform.

[1] High Productivity Computing Systems (HPCS) Program, http://www.darpa.mil/ipto/programs/hpcs/

3 TOOL INTEGRATION

As any tool developer knows, a substantial amount of effort is required to establish the infrastructure needed for the tool to operate, port and maintain the tool for different operating systems and architectures, and develop the user interface needed to interact with the tool. In addition, for separate tools to interoperate in any meaningful way, a range of tool-specific services, APIs and frameworks must be developed and maintained. The trade-off between the cost of these activities and the investment in the tool itself has lead to the current situation where very few integrated tools exist.

Eclipse significantly reduces this cost by providing a central integration point that allows tools to integrate with a platform rather than each other, and by supplying a large number of services, APIs and frameworks that can be leveraged by the tool instead of needing to be developed individually.

Although the ultimate goal of the integration strategy is to achieve a development environment where all tools are fully integrated, the investment in existing tools means this clearly cannot happen immediately. Migration of existing tools and tool development projects to Eclipse is an important part of the strategy, since it is necessary to ensure that the current developer tool requirements continue to be met.

Eclipse makes it possible to employ an incremental tool integration process that will ensure that the integration of existing tools happens in a manageable way. There are four main levels of tool integration in Eclipse [2]:

- *Invocation* – this is the minimum level of integration allowing the tool to be accessed from the IDE, but without using common services or data formats. Minimal integration requires that a tool can be built and installed through the IDE resulting in linkable libraries or applications that can run through the builder tool chain[1] (like compilers currently are).
- *Data* – this allows tools to share data by defining standard mechanisms for accessing and exchanging data. This is generally independent of tool specific data repositories, but provides a means of accessing data stored in these repositories. Eclipse also provides a means of persisting tool state information as resources that can be accesses using standard mechanisms.
- *API* – this allows client applications to access tool-specific functions through well defined APIs. Eclipse allows these APIs to be exposed using a plug-in manifest file and enables tool functionality to be re-used or extended through the use of extensions and extension points. This would allow, for example, source-to-source transformations tools to be integrated with the editor, tool chains, and debugger.
- *User Interface* – this level of integration allows tools to interact as if they were designed as part of a single application. Tools utilize Eclipse platform extension points to re-use viewers, editors, and other functionality. Tools are able to register

[1] A tool chain is a simple XML description of the tools and tool products used to build an application.

interest in other tools so they are notified when events occur. For example, the parallel debugger requires complex interactions between the native back-end debuggers, the run-time system, the source editors, and other user interface components.

Existing tools will invariably begin by being integrated at the *invocation* level since this requires few changes to the existing tool and the least investment in resources. Once this level of integration is achieved, the tool developer can explore opportunities for providing *data* and *API* integration. Finally the tool can be fully integrated by moving the user interface over to the Eclipse platform. This will normally require re-implementing a significant portion of the tool.

In some cases it may make sense for a tool to remain only partially integrated, or integration achieved through the use of an external interface. Examples of this include compilers and other tools that are closely linked to system infrastructure. Eclipse provides a number of mechanisms to support this type of integration.

4 LANGUAGE INTEGRATION

The integrated development environment should support existing languages used in developing scientific applications such as Python, C, C++, and Fortran as well as interesting new parallel languages such as Co-Array Fortran (currently in the standardization process) and UPC.

Because a full-featured IDE requires complete program parsing (with either internal or external tools), it can become an effective platform for static analysis and tools employing source-to-source transformations, especially when integrated with the build system and the debugger. For example, the build system can be configured to automatically create transformed files (as the application is built) and the debugger (coupled with the editor) can display program counters in either the original or transformed files.

An integrated platform for static analysis can also provide common interfaces for many tools that require source-to-source transformations, including, code refactoring (e.g., MPI data-type extraction), code optimization (e.g., loop unrolling), automatic differentiation[1], and performance monitoring (e.g., insertion of trace calls).

Like tool integration, there are a number of levels of language integration in Eclipse. The four main levels are:

- *Build & Compile* – this level is concerned with how the program is compiled and built, and these two activities tend to be closely linked. The options are to use an external builder (for example the "make" program) or utilize the internal Eclipse build system. Using an external builder will normally necessitate using an

[1] Automatic differentiation, http://www-unix.mcs.anl.gov/~naumann/nice_mini.html

external compiler. For greater integration, it is also possible to write the compiler in Java and integrate it directly in the Eclipse platform.

- *Launch* – although not generally an issue for traditional languages (since the program runs directly on the local machine), program launch becomes a significant factor for integrating parallel languages. This is because in most cases the program will not be running locally (except for some shared memory type architectures) and normally requires support from an external runtime system.

- *Language Model* – the internal language model (known as the document object model or DOM) is an important part of the language integration process. The DOM is used for indexing of program elements for display in an outline view, and complete parsing of program structure for static analysis and refactoring. In addition, the DOM can provide language specific information to other tools such as the debugger.

- *Language Specific Editor* – the editor is an important part of the language integration process, and plays a large part in defining the language for the developer. The editor provides features such as keyword highlighting, code formatting, syntax checking, content assist and refactoring.

The following table provides a snapshot of current languages already integrated into Eclipse.

Language	*Build/Compile*	*Launch*	*Model*	*Editor*
Java	internal/internal	yes	yes	yes
C/C++	external/external[1]	yes	yes	yes
Fortran	external/external[1]	yes	yes	yes
COBOL	external/external	yes	no	yes
Python	external/external	yes	no	yes
Perl	external/external	yes	no	yes
Ruby	external/external	yes	no	yes
Haskell	external/external	yes	yes	yes
Eiffel	external/external	yes	no	yes

This shows that build and compile support is most easily provided using external infrastructure, and that launch (onto a local machine) and language specific editor support is also relatively easy to provide. Integrating the language with the internal language model is normally the most difficult part of the integration process, as it generally requires rewriting the compiler front-end. Future work planned for PTP will simplify this process by allowing an externally generated intermediate representation or API to be used to populate the DOM.

[1] Also provides a managed build system

5 INTEGRATION EXAMPLES

We emphasize that the Eclipse PTP is a platform for tool integration, not just another set of new tools. This is demonstrated by three concrete examples of how the PTP can be (or has been) used to integrate the class of tools utilizing source-to-source transformations.

5.1 TAU

The TAU (Tuning and Analysis Utilities) project is a set of tools that gather performance information through instrumentation (source-to-source transformations) of user code. This includes tools for instrumentation, measurement and analysis of application performance. Opportunities for Eclipse integration exist for all these tools.

Initial integration of TAU with Eclipse and the PTP at the *invocation* level has already been accomplished. The TAU compiler has been integrated with the Eclipse builder tool chain to allow source code instrumentation to occur as part of the normal build process. Once the application has been launched and profiling data collected, the TAU ParaProf profile visualization tool can then be launched from Eclipse to view and manipulate the profile data.

For *data* level integration, TAU's profiling database infrastructure PerfDMF could also be integrated with Eclipse. This should be relatively simple, as PerfDMF already provides a Java API for database query and analysis. Since PerfDMF offers the ability to import performance profile data from other tools into its performance database, this would allow PerfDMF to serve as a performance data integration system for PTP. New analysis tools could then be developed that work with the PerfDMF data model, rather than requiring a new set of instrumentation and measurement tools to be integrated with the PTP each time.

TAU provides a rich set of APIs so there should be ample opportunity for exposing these in the *API* level integration. In addition to exposing the PerfDMF API, it may also be possible to expose aspects of the measurement API to other tools. In addition, this would be a good opportunity to provide a well defined interface to the performance analysis operations.

The final *user interface* level of integration would involve extending and developing a number of user interface components. This would range from using the existing Eclipse resource explorer to replace equivalent functionality in TAU, providing enhancements to the editors to display and manipulate instrumentation and profiling information, and converting ParaProf views to use Eclipse infrastructure. This level of integration would allow PTP to link program information with the events the performance tool is instrumenting and measuring, and with post-execution analysis. This would make available significantly more semantic context, relative to the program, than is usually provided by a standalone performance tool.

5.2 ROSE

ROSE [14] utilizes compilation techniques to address the optimization of user-defined abstractions. Like TAU, it also transforms user code, but for purposes of code optimization or code refactoring. ROSE works by generating an abstract syntax tree (AST) representation of the source code, transforming the AST using the user-defined abstractions, and then un-parsing the AST back into optimized source code. At this point, the optimized source code can be compiled to executable form using a conventional compiler.

ROSE is an example of a tool that would only be partially integrated into Eclipse. Since it utilizes proprietary components, the operation of the tool will need to remain external to the Eclipse platform. However, there are still a number of areas where integration could be usefully undertaken.

At the *invocation* level, it would be possible to incorporate ROSE into the normal compiler tool chain so that optimizations could be carried out automatically as part of the build process. In addition, launching an external viewer could support visualization of the generated AST. This allows the developer to better understand the transformations taking place.

At the *data* and *API* levels, an interface to the AST could be exposed to Eclipse-based tools. This would allow other tools to access and traverse the generated AST. Such an interface may also allow the AST to be used to populate the internal language models so that existing refactoring operations could be performed.

Since ROSE has no user interface of its own, there is ample opportunity to provide *user interface* level integration. By integrating with Eclipse menus and editors, ROSE could be used to provide access to a range of optimizations and refactorings as code is being developed. For example, an "Add Implicit None" refactoring could be selected during the course of editing a Fortran 77 file. This would automatically run an existing ROSE transformation (add_implicit_none) on the file and provide transformed source code back to Eclipse, where it would be displayed by the editor. The un-transformed and the transformed source code could also be compared side-by-side using the built-in difference viewer.

5.3 CO-ARRAY FORTRAN

Co-Array Fortran (CAF) is a simple parallel extension to the Fortran language that allows memory residing on another processor to be directly addressed. The Rice Co-Array Fortran compiler [4] transforms CAF programs to Fortran 90, which are then compiled with the user's normal Fortran compiler. CAF provides an example of language integration in Eclipse.

Like other compiled languages, the *build & compile* integration of CAF is most easily achieved by extending the existing Fortran tool chain to invoke the CAF compiler prior to the Fortran compiler.

Providing *launch* integration is more difficult, since CAF relies on calling either the ARMCI [1] or GASNet [8] communication libraries. However extending the parallel launch capability of PTP to support communication primitives for SPMD languages will solve this problem. This extension is planned for a future release of PTP.

Because CAF is only a small set of language extensions to Fortran 90, providing *language model* integration should be relatively simple. The existing Fortran parser and AST, that are provided as part of the Eclipse Photran project, can be extended to include the new language features. Some changes to the static analysis and refactoring tools would also be required.

Similarly, the *language specific editor* level of integration could be achieved by extending the existing Fortran editor and outline views provided by the Photran project.

6 CONCLUSION

The advent of the Eclipse platform and the establishment of the Parallel Tools Platform project have provided a unique opportunity to solve some of the major challenges that are facing the effective realization of the current HPCS strategy.

The need for such an approach has never been more apparent. The order of magnitude improvement in productivity that programs such as HPCS are promising to deliver will not be met by hardware and system software alone. If such goals are to be achieved, then a software infrastructure that improves the development process and usability of these machines is going to play a key role.

Without such a parallel tools strategy and related integration effort, it is highly likely that parallel tool development will proceed in the same ad-hoc manner that it does now, with the consequence that it will become more and more difficult to reach the productivity goals demanded by future HPCS systems.

REFERENCES

[1] Aggregate Remote Memory Copy Interface, http://www.emsl.pnl.gov/docs/parsoft/armci/

[2] Amsden, J., "Levels Of Integration: Five ways you can integrate with the Eclipse Platform", http://eclipse.org/articles/Article-Levels-Of-Integration/Levels%20Of%20Integration.html, March 2001

[3] Cleanscape Grayboxx, http://www.cleanscape.net/products/grayboxx/index.html

[4] Co-Array Fortran effort at Rice University, http://lacsi.rice.edu/software/caf/

[5] Collette, M., Corey, B., and Johnson, J., "High Performance Tools and Technologies", tech report, Lawrence Livermore National Laboratory, UCRL-TR-209289, December 2004

[6] El-Ghazawi, T., et. al, "UPC: Distributed Shared Memory Programming", ISBN 0-471-22048-5, John Wiley and Sons, May, 2005

[7] Ganglia, http://ganglia.sourceforge.net/

[8] GASNet, http://gasnet.cs.berkeley.edu/

[9] HPC Toolkit, http://freshmeat.net/projects/hpctoolkit/

[10] Moreira, J.E., et. al., "Treating a user-defined parallel library as a domain-specific language", Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS 2002, pp 105, April 2002

[11] Numrich , Reid, "Co-Array Fortran for Parallel Programming", ACM Fortran Forum, vol 17, no 2, pp 1-31, 1998

[12] Open|SpeedShop, http://oss.sgi.com/projects/openspeedshop/

[13] Platform LSF, http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/

[14] ROSE, http://www.llnl.gov/CASC/rose/

[15] Supermon, http://supermon.sourceforge.net/

[16] TAU - Tuning and Analysis Utilities, http://www.cs.uoregon.edu/research/tau/home.php

[17] Van De Vanter, M.L., Post, D.E., Zosel, M.E., "HPC Needs a Tool Strategy", Proceedings of the Second International Workshop On Software Engineering For High Performance Computing System Applications, May 2005, pp. 55-59