Improving UPC Productivity via Integrated Development Tools

Max T. Billingsley III
University of Florida
billingsley@hcs.ufl.edu

Beth R. Tibbitts
IBM Corporation
tibbitts@us.ibm.com

Alan D. George
University of Florida
george@hcs.ufl.edu

Abstract

In the world of high-performance computing (HPC), there has been an increased focus in recent years upon the importance of productivity in HPC application development. One crucial aspect of productivity is the programming model used, and the family of partitioned global-address-space (PGAS) models, such as UPC and X10, has served to advance the state of the art in balancing performance and productivity. Also of great importance is the variety of development tools used to support activities such as editing, debugging, and optimizing programs. These tools are often most useful as part of an integrated development environment (IDE). While some progress has been made towards bringing IDE capabilities into the HPC world, in particular by way of Eclipse projects, support has mainly focused on MPI and OpenMP tools.

In this paper, we present research and development activities that are bringing Eclipse-based IDE capabilities to the PGAS developer community. We focus on tools for UPC, giving background on previously existing capabilities to work with UPC programs in Eclipse and then presenting a tool-chain and project wizard for the open-source Berkeley UPC compiler, basic UPC static analysis tools, and integration of our performance analysis tool (Parallel Performance Wizard) supporting UPC. Finally, we conclude by proposing future work and providing recommendations for further integration of UPC and other PGAS tools to enhance overall developer productivity.

Categories and Subject Descriptors D.2.6 [*Software Engineering*]: Programming Environments—Integrated environments; D.1.3 [*Programming Techniques*]: Concurrent Programming—Parallel programming

Keywords Eclipse, partitioned global address space, UPC

1. Introduction

The field of high-performance computing (HPC) has as its explicit goal the achievement of high levels of application performance, allowing ever-larger and more complex problems to be solved. Parallel computing has long been the dominant paradigm in HPC. There are a diversity of parallel systems in use, ranging from smaller shared-memory systems (notably including modern multi-core workstations) to commodity clusters (with interconnects ranging from Ethernet to high-bandwidth, low-latency networks such as InfiniBand) to high-end offerings such as Cray XT and BlueGene systems. To exploit the potential these systems offer, programmers must cope with the many complexities of the machines and their software environments, and they need effective tools to craft programs that take advantage of the capabilities of highly parallel HPC systems.

One key aspect is the programming model used. For over a decade, MPI [1] has dominated the HPC development landscape, with OpenMP [2] often being used for finer-grained coordination within an HPC system node. But MPI has some notable drawbacks, specifically with regard to ease of use (in particular because programmers must manually coordinate send/receive pairs) and overall programmer productivity.

For many years, efforts have been under way to develop new parallel programming models that would be a real improvement over MPI (and OpenMP). One particular effort has centered on the family of partitioned, global-address-space (PGAS) programming models, which promise an improved balance of productivity and performance by way of their shared-memory abstraction and explicit notion of data affinity. Probably the most mature member of the PGAS family is the UPC language [3]; others include Titanium [4] and the newer X10 [5] and Chapel [6] languages being developed as part of the DARPA High Productivity Computing Systems (HPCS) initiative [7].

The HPCS program has emphasized the importance of productivity in the HPC world. While sheer application performance is clearly a vital aspect of HPC, what application writers need really be concerned with is total-time-to-solution, which includes overall development time as well as application run time. Previous work [8] has identified time-to-solution as the key productivity metric for programming

languages (along with other development tools). Development time is of course directly impacted by developer productivity, a measure of how quickly and efficiently the developer can turn the application requirements into a complete, correct, and efficient program. While there is existing work toward precise metrics for programmer productivity [9], we do not concern ourselves in this paper with analytical or emperical methods to analyze productivity, but rather accept the general notion that developers seek to produce a correct, performant program as quickly and easily as possible.

There are a number of other tools that may aid productivity during the many stages of application development. These include source-code editors, debuggers, profiling tools (e.g., gprof), and a wide variety of other static and dynamic analysis tools. For many users, these tools may be most useful when they are part of an integrated development environment, or IDE. One integration platform of particular interest—because of its extensible architecture, open-source licensing, and growing developer community—is Eclipse, which will be described in further detail below.

In this paper we present a range of UPC application development tools as integrated with the Eclipse platform to comprise an integrated development environment for UPC. The remainder of the paper is organized as follows: Section 2 gives an overview of the Eclipse platform and architecture and outlines requirements for UPC support in Eclipse; Section 3 then covers UPC tools in Eclipse. We present the integration of our parallel performance tool which supports UPC in Section 4, before presenting a brief application study in Section 5, and finally concluding and discussing future work in Section 6.

2. Eclipse Platform and Architecture

Eclipse [10] is an extensible, open-source platform for integrating software development tools. From a user's point of view, Eclipse provides a powerful integrated development environment (IDE) that can help the user be productive throughout each stage of software development. Architecturally, Eclipse comprises an general-purpose tool-integration framework in which a wide variety of tools can be added in the form of Eclipse plug-ins.

Eclipse was originally created by IBM as a Java IDE, but is now governed by the Eclipse Foundation [11]. Numerous projects have since added support for many other programming paradigms and languages. One such project is Eclipse CDT (C/C++ Development Tooling) [12], which provides a full-featured environment for developing programs written in C, C++, and related languages. Because much HPC (and other parallel) development is done using languages in the C/C++ family, CDT forms an important part of the Eclipse ecosystem that has particular relevance for a parallel development environment.

A more recent but no less important part of the Eclipse ecosystem is the Eclipse PTP (Parallel Tools Platform) project [13], which is aimed at bringing integrated development capabilities to the world of parallel application development. Previous work has shown that PTP and associated tools can offer substantial benefits for the parallel development process [14]. Some of the particular capabilities offered by PTP include: a variety of error checking and static analysis tools; runtime monitoring and control of parallel jobs; debugging support for multi-process applications; and an external tools framework for integrating dynamic analysis tools. The PTP project initially focused mainly on MPI and OpenMP development tools.

The remainder of this paper presents recent and current work to bring a parallel development environment to the world of PGAS, beginning with UPC. This work leverages and extends capabilities of the Eclipse platform, CDT, and PTP to comprise an Eclipse-based integrated toolset for UPC application developers.

Requirements for UPC Support

In order for Eclipse to be a useful platform for UPC developers, there are a number of requirements that must be fulfilled. These requirements fall in a few distinct categories: basic support for the UPC language; capabilities related to building a UPC application; functionality for running the resulting program; and support for other development steps such as debugging and the use of other static and dynamic tools. Here we address further considerations for each of these categories of requirements.

Perhaps the most obvious of the requirements is basic support for the UPC language itself; this support would entail handling UPC as a top-level source program language, identifying UPC with the .upc (and optionally .c) filename extension. Recognizing the UPC language is important not only for basic tools such as the source-code editor but also so that any other tool plug-ins can identify UPC as the applicable language and function accordingly.

Another requirement for UPC support is the availability of one or more compiler plug-ins so an application can be built within the Eclipse environment. Users need to be able to specify any relevant UPC compilation options and see the output of programs invoked during the build process. The build process with a supported UPC compiler should be smoothly integrated into the Eclipse workflow, ideally with most or all of the tedious details associated with the application build handled by the system.

The requirements for running a UPC application are generally quite similar to those for running other parallel applications, in particular those written using another programming model with the single-program-multiple-data (SPMD) execution model, such as MPI. The PTP project generally provides the necessary capabilities by way of the available resource-manager plug-ins and parallel runtime system.

Other development steps that should also be covered by UPC tools in Eclipse include debugging, analyzing the performance of an application, and various other static and dynamic analyses. However, it should be noted that in the case of debugging, there is currently no freely available, open-source debugger that fully supports UPC. Providing such a debugger should be considered an initial requirement that will hopefully soon be met by the wider UPC community. At that point the parallel debug platform provided by PTP will serve as an ideal platform for integrating a UPC debugger.

Challenges

There are a number of challenges associated with providing UPC support in the Eclipse environment. The overarching challenge is to provide sufficient functionality, with compelling features and usability, such that UPC developers will consider adopting the environment. Additionally, this functionality needs to be cleanly integrated within Eclipse, which itself is a challenge because of the scope and complexity of the Eclipse infrastructure. This integration is, however, well worth the effort required, because of the substantial existing Eclipse functionality that becomes available for UPC developers. One final challenge arises from the fact that many required Eclipse components were not initially designed with UPC (or other PGAS languages) in mind; a number of these components (e.g., the External Tools Framework in PTP, which will be mentioned later in the paper) required at least minor modifications to support UPC.

3. UPC Tools in Eclipse

In this section we cover the variety of UPC-related capabilities that have been added to Eclipse. We first cover some existing functionality in the CDT project which gives basic support for the UPC language. We then present our work on a fully functional plug-in for the Berkeley UPC compiler and basic static analysis tools for UPC.

Some existing basic support for the UPC language is provided by the CDT project, which has added UPC to the family of C/C++ languages handled by CDT. The original parser as part of CDT was hand-coded for the particular syntax of GNU C. An additional, more extensible parser was written by IBM and employs an externally-generated grammar file for the particular language variation. This capability was used by the UPC tools to provide a UPC-specific parser that the C editor employs to recognize the particulars of the UPC syntax. This enables usage of parsing- and indexing-based features such as content assist, navigation, search, call hierarchy, and type hierarchy, along with UPC-aware syntax highlighting in the source-code editor itself. A screenshot which shows a UPC program in the Eclipse editor will be shown in Section 5 below.

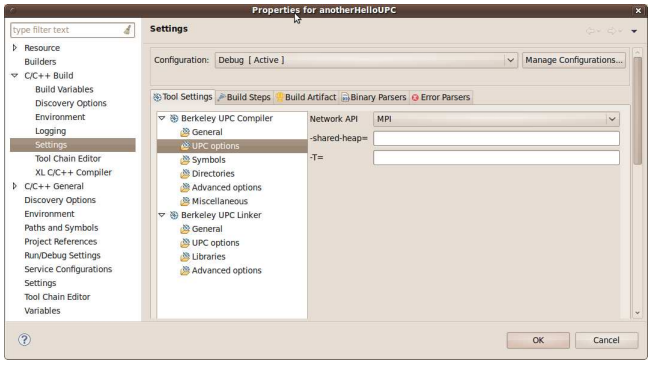

Figure 1. Dialog for Berkeley UPC settings.

3.1 UPC Compiler Toolchains

One of the most important aspects of UPC support in Eclipse is integration of a UPC compiler available to the user. In the Eclipse world this is achieved by creating a plug-in for CDT that provides a new *tool-chain* for the compiler to integrate it with CDT's managed-make system (in which CDT manages the build process instead of requiring users to supply hand-written Makefiles). The tool-chain plug-in encapsulates the means of properly invoking the compiler and linker, and provides access to compiler and linker options by way of a standard CDT user interface.

Our contribution in this area has centered on adding support for the open-source Berkeley UPC [15] compiler. Our tool-chain plug-in includes all of the aforementioned functionality, giving full access to the compiler within Eclipse. Because there are many possible options for the Berkeley UPC compiler, all of which can be important for the UPC programmer, our toolchain plug-in provides an interface to specify Berkeley UPC options within Eclipse. This interface integrates with the standard build-settings interface provided by CDT. When the Berkeley UPC tool-chain has been selected, choosing **C/C++ Build→ Settings** in the project properties dialog brings access to the interface shown in Figure 1.

CDT also includes another UPC toolchain provided for the IBM UPC compiler [18]. The IBM xlUPC toolchain integrates with the managed-build system and provides a user interface to set compiler options in a user-friendly way. Additionally, the xlUPC toolchain is remote-enabled via the PTP Remote Development Tools (RDT). RDT allows CDT projects to exist on a remote machine; such projects can be built and launched on a remote machine as well. This is well-suited for development of large code bases on large remote clusters. The UPC parser is remote-enabled to facilitate remote parsing and indexing of UPC source code, allowing UPC projects to be efficiently developed remotely. A remote xlUPC toolchain is also available to facilitate remote builds of the xlUPC project. The UI provides a user-friendly way to specify and manage complex build options without makefiles.

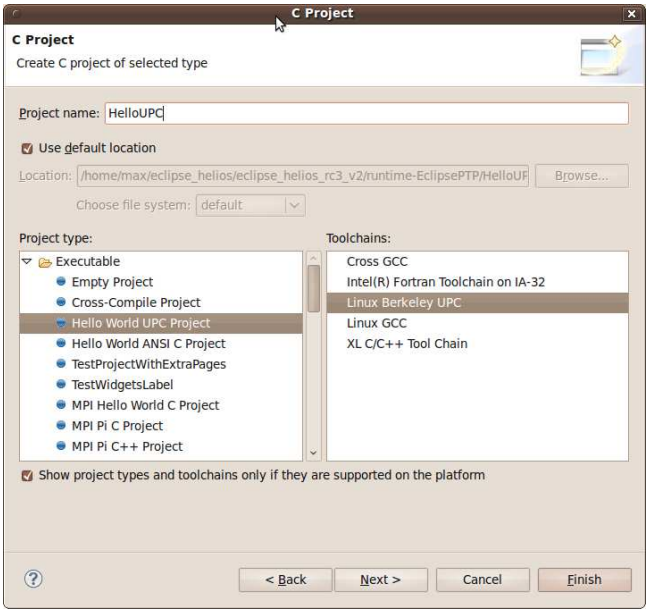

Figure 2. First page of UPC project wizard.

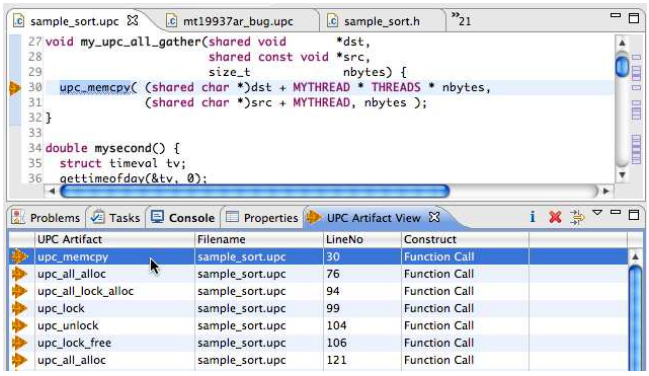

Figure 3. UPC artifacts are identified and linked with editor.

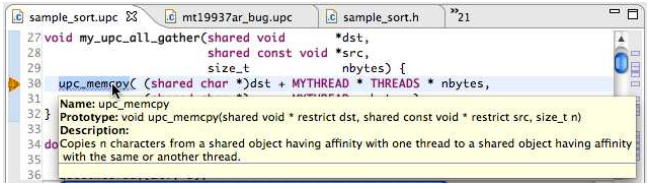

Figure 4. Eclipse hover help gives API assistance for UPC APIs and keywords.

3.2 UPC Project Wizard

One important part of an IDE is a way for users to quickly and easily create new projects using a given programming model. In general, the notion of *projects* in Eclipse (as well as other IDEs) is important for organizing and managing the components of (and settings for) an application over the course of its lifetime. Similarly, project templates can be helpful for new and experienced programmers alike, as they simplify and accelerate the routine process of beginning a development project. To address this need for UPC programmers, our Berkeley UPC tool-chain plug-in provides the new *UPC Hello World Project* type. When a user chooses to create a new UPC project, they can now select this project type (along with the Berkeley UPC toolchain) in the dialog shown in Figure 2.

When a project is created using the *UPC Hello World Project* type, the appropriate directory structure and several files are created within the Eclipse workspace. One of these files is an initial UPC source file, which contains code for a simple "hello world" UPC program. The project can immediately be built using Berkeley UPC, giving users a simple, quick onramp to a working UPC application. Throughout the course of a UPC application's lifetime, the project abstraction in Eclipse helps developers keep track of all associated components and metadata.

3.3 Analysis Tools in PTP

PTP includes several static analysis tools as part of the Parallel Language Development Tools (PLDT) feature of PTP. These tools generally aim at performing analyses based solely on the program source code (along with the syntax and semantics of the applicable programming model) to inform the programmer of any potential problems with or opportunities for improvement of the application.

As an initial step toward static analysis functionality for UPC, we have created functionality allowing UPC "artifacts," such as calls to UPC library APIs, to be located and displayed in a separate Eclipse view, with easy navigation to source-code lines. The UPC Artifacts view is shown in Figure 3; this view is sortable by file, line number, or artifact and also links directly to the Eclipse editor.

For MPI and OpenMP, more sophisticated analysis tools include MPI barrier analysis that detects deadlocks, among other features; MPI and OpenMP artifacts can also be located. The infrastructure exists for more advanced analysis tools for UPC. The parallel analysis features use the existing CDT AST (Abstract Syntax Tree) and also a PLDT-provided control-flow graph and other structures that allow for analysis of complex program structures. These are currently used for various MPI and OpenMP analysis features—and the opportunity exists for more complex analyses of UPC code as well.

3.4 User Assistance Tools for UPC

The Eclipse help system also provides hooks for dynamic help features. Hovering over the UPC APIs in the editor invokes a "pop-up" feature with information about the API, sourced both from header files and from additional help information explicitly provided by PTP and PLDT. The hover help for UPC is shown in Figure 4.

There is also a separate help view available via the F1 key or **Help** → **Dynamic help** menu, and small HTML

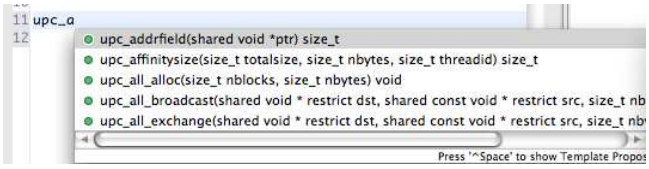

Figure 5. Eclipse content assist inserts code into the editor as you type (a popup describing the API selected is not shown here).

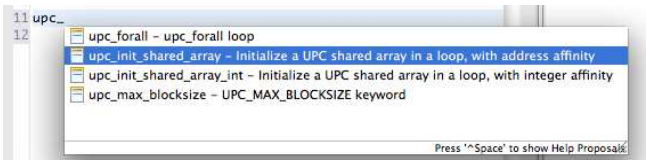

Figure 6. Editor templates ease coding of common idioms.

files can be perused which provide information on the APIs and, if provided, examples of their use. This and the hover help described above makes information normally found in reference books immediately available at the user's fingertips during code development. Additionally, the CDT parser recognizes incorrect language usage and immediately displays compiler error messages in the *Problems* view while also annotating the editor with error markers on the lines containing the errors.

Content assist in the Eclipse editor helps remind the programmer of language/library APIs and their arguments; this assistance occurs dynamically as the user types source code. When the user enters a few characters of the name of an API routine, for example, and then presses the Control-Space key combination, a popup appears with choices and help information about the APIs available. This capability is shown for UPC in Figure 5. If the user chooses one of the API choices provided with the mouse or arrow keys, the selection is directly inserted in the code at the cursor position. Hover help follows along as the user types arguments as a reminder of required argument types. Because UPC consists of not only language features but also extensive library functionality, this type of dynamic content assist is impactful for improving UPC developer productivity during source-code editing.

Code templates also make programming of language idioms or other commonly used programming constructs easy to implement. For example, a *upc_forall* loop with shared-address or integer affinity is provided as one of the built-in templates provided by PLDT, and can easily initialize a UPC shared array in the loop. Figure 6 illustrates this capability. Variable parts of the template can be quickly modified to use variables within the user's own program. Addtionally, more code templates are easily added by the user in the Eclipse preferences.

These and many other features help the programmer increase productivity while working on a UPC application. The descriptions here are for UPC-specific features, but many other features help with code formatting, indenting, comparisons between versions of a file (e.g., from a CVS or SVN repository, etc.). Compiler error messages map easily to source-code lines to easily identify and correct problems. While this type of functionality is quite widely used in the Java community, for example, these productivity-related capabilities represent a step forward for a parallel language like UPC.

4. Parallel Performance Wizard Support

In this section we present a key contribution of this paper, namely the integration into Eclipse PTP of our performance analysis tool, called Parallel Performance Wizard (PPW), which is described in detail in [16]. This integration serves as a prime example of the usefulness of Eclipse PTP as a platform for bringing together a variety of PGAS tools.

4.1 PPW Overview

Once a developer has completed a working parallel application, an experimental performance-analysis process is often needed in order to optimize application performance. This process often takes the form of a measure-modify cycle, in which an instrumented version of the program is executed, performance data is collected at runtime, and the results are examined by the programmer and used to modify the program to improve its performance. Performance tools help with each step of this process, facilitating program instrumentation, performing measurement and handling of performance data, and then presenting the data via user-friendly visualizations.

PPW is an experimental performance tool which focuses on optimization of applications written using PGAS programming models.

4.2 PPW Integration

PPW has been integrated into Eclipse by way of PTP's External Tools Framework (ETFw), which is designed to facilitate integration of existing parallel tools (performance analysis tools being one key example) into the Eclipse world. The ETFw defines a number of integration points for which a tool specifies certain behavior to occur. The following list shows each integration point and the corresponding behavior specified by our PPW plug-in for each:

- *Instrumentation.* Nothing need be performed in this step (instrumentation is handled by the next step).
- *Build.* In place of the usual UPC compiler invocation, the PPW compiler wrapper script is invoked to perform an instrumented build of the application, with options corresponding to those specified by the user.
- *Launch.* A parallel launch is performed using PTP's parallel runtime, with special PPW environment variables set to pass along options specified by the user.

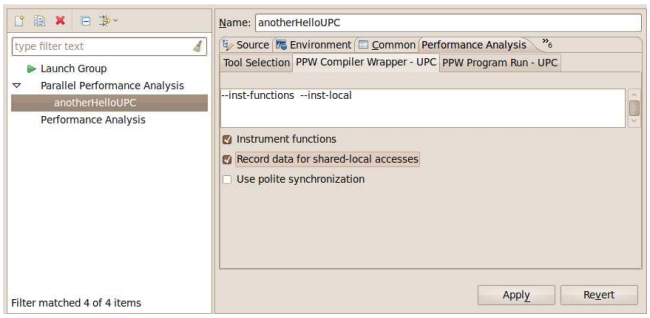

Figure 7. Dialog to specify instrumented-build options for PPW.

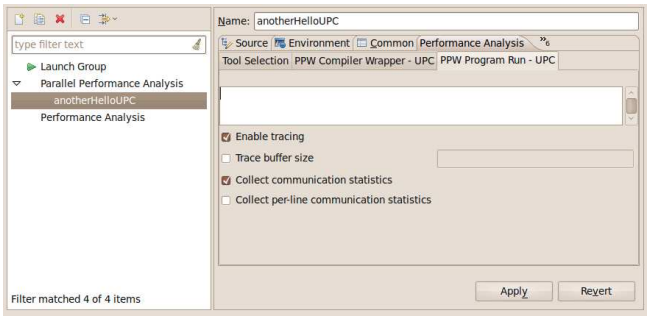

Figure 8. Dialog to specify program-run options for PPW.

- *Data Management.* The resulting performance data file is placed in the user's workspace and named appropriately (based on the current Eclipse project name).
- *Visualization.* The PPW user interface is invoked in a special mode for interaction with Eclipse; the performance data file is opened.

The behaviors specified above combine to provide access to essentially all of PPW's capabilities from within the Eclipse environment. Users make use of these capabilities by creating a profile configuration for profiling the application using PPW. Just as run configurations can be created to perform a parallel application run using PTP, profile configurations encapsulate all necessary setup to build an application with performance analysis support and then perform instrumented program runs to collect performance data.

Users simply select **Run → Profile Configurations...** to begin setting up a profile configuration for their application, and then choose **Parallel Performance Analysis** to create a new profile configuration. Many of the fields are the same as for a corresponding run configuration, but a few important new tabs will be available in the **Profile Configurations** dialog. Selecting the **Performance Analysis** tab allows the user to choose PPW from the available performance tools. To set PPW-specific options, users then use the **PPW Compiler Wrapper - UPC** tab (Figure 7) to specify various options for the instrumented build with PPW. Similarly, the **PPW Program Run - UPC** tab (Figure 8) is used to specify options for the parallel program run with PPW.

After the program is launched with PPW support enabled from within Eclipse, the resulting performance data file is placed within the user's workspace. The PPW user interface is then automatically launched in a new special mode allowing for interaction with the Eclipse environment. One primary feature of this mode is that source-code locations corresponding to performance data being viewed within PPW are shown directly in the Eclipse editor. An example of such a view, showing the PPW GUI and the Eclipse editor window, is show in Figure 9.

5. Application Study

In this section we describe a brief application study which serves as an example for using the various UPC capabilities available within Eclipse.

5.1 Application Overview

The application chosen for this study is a UPC implementation of the Synthetic Aperture Radar (SAR) algorithm completed by a fellow researcher in our lab; the UPC version was in turn based on the sequential version of SAR from the Scripps Institution of Oceanography and MPI versions provided by two additional researchers in our lab [17]. With the MPI version as a starting point, several different UPC versions were created; in this paper we will primarily work with the initial UPC version (version 1) of the SAR application.

5.2 Project Setup

To begin working with the SAR application, we first opened Eclipse, switched to the C/C++ perspective, and created a new Empty C/C++ project corresponding to SAR version 1. Upon creating the project, we chose our Berkeley UPC toolchain (described in Section 3.2) to use for building. Because we already had existing source code for the application, we then used the import function of Eclipse (by right-clicking on the project to bring up the **Import...** menu) to bring the code into the Eclipse project. Note that if this had been a completely new program, we could have instead created a source directory and new UPC source files (using a simple wizard in Eclipse) to begin working. Figure 10 shows the SAR application within the Eclipse workspace, with the source-code editor opened to the main UPC file. The program consists of a number of source files; a view of the source tree can be seen in the leftmost pane within Figure 10.

5.3 Build and Run

To obtain an appropriate setup to build our application, we used the (previously described) Berkeley UPC dialogs to adjust compilation options; for example, we added several libraries needed for the link step. Adding such libraries for the linker is particularly straightforward within Eclipse

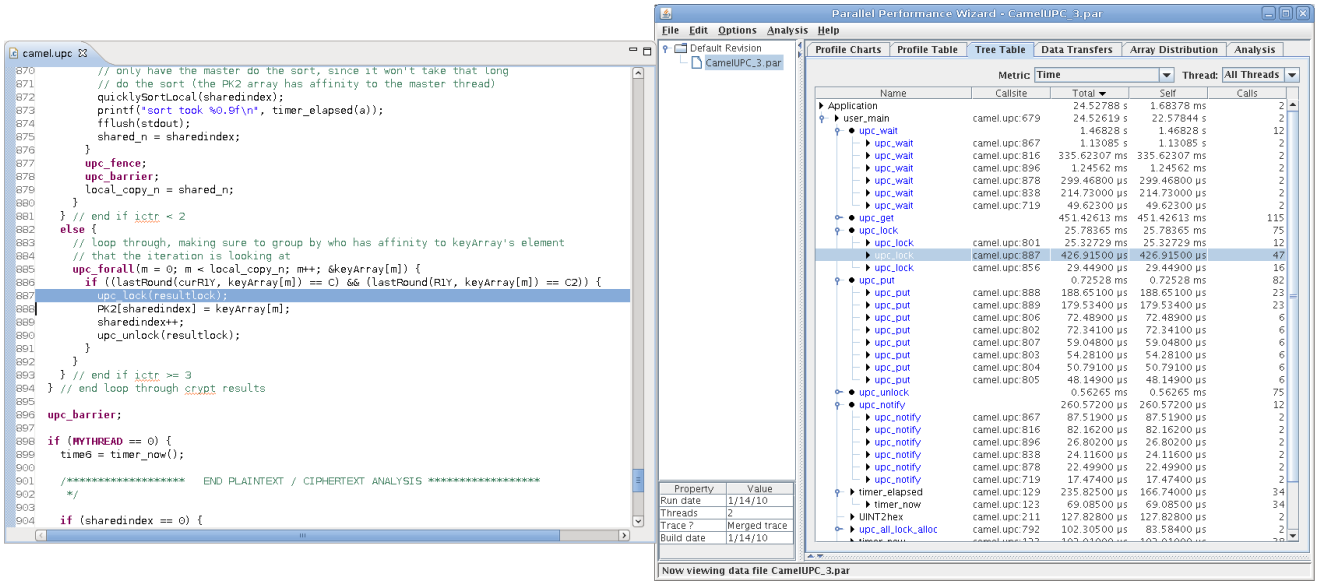

Figure 9. PPW GUI and Eclipse editor showing source-code correlation.

because of the convenient interface provided by way of the CDT project. We were then ready to build our application—this was as simple as clicking on the 'Build' button in the main Eclipse view. Once the build itself was invoked, we were able to continue working within Eclipse while the build progress was depicted in the *Progress* view. The *Console* view provided a listing of the output from Berkeley UPC; this can be seen in the bottom portion of Figure 10.

Once the build completes, the resulting executable is placed within the project workspace (as can be seen listed in the left portion of Figure 10. To perform a parallel run of the application, we first switched to the PTP *Parallel Runtime* perspective and created a new resource manager corresponding to our execution environment. For the sake of simplicity, we used a basic OpenMPI resource manager to run the application on the 16-core SMP system on which we were developing the SAR application (and on which we were running Eclipse).

Once the resource manager was created and started, we selected **Run → Run Configurations...** to create a new *Parallel Application* run configuration for the SAR application. Here we chose the resource manager created above, selected the program executable (from within the project workspace), and specified the number of processes to use for the run along with all options to be passed to the application executable. Once the run configuration was complete, we simply clicked "Run" to begin the execution. While the program was running we were able to view the status of the job and its processes within the *Jobs List* and *Machines* views in the PTP *Parallel Runtime* perspective, and console output from the program could be seen in the *Console* view.

5.4 Performance Analysis

In order to conduct performance-analysis experiments with the SAR application using PPW, we had one initial configuration step to perform, whereby we specified the location of our PPW installation. This field was accessible via the dialog under **Preferences → Parallel Tools → External Tools → Tool Location Configuration**. We then created a profile configuration with many options set the same as for the run configuration previously used. We decided to perform a full program trace of both UPC operations and our own user functions, so we set the PPW build and run options accordingly via the dialogs described in Section 3.2 above. We then started the run; while waiting for it to complete, we were again able to monitor its progress via the *Parallel Runtime* perspective. We also continued to use the Eclipse editor to examine the SAR source code itself while the run was taking place.

Upon completion of the run, the PPW GUI was launched with the results from our performance experiment. Examining the resulting data allowed us to observe the main performance problem with the initial SAR code (which our colleague had previously identified): namely a poorly balanced synchronization relating to I/O performed solely on a master UPC thread.

6. Conclusions and Future Work

This paper has presented our work towards an integrated development environment for UPC that aims to substantially improve developer productivity throughout the many stages of HPC application development. After giving an overview of the Eclipse platform and architecture and outlining requirements for UPC support in Eclipse, we presented a range of UPC tools as integrated into Eclipse. Specifically,

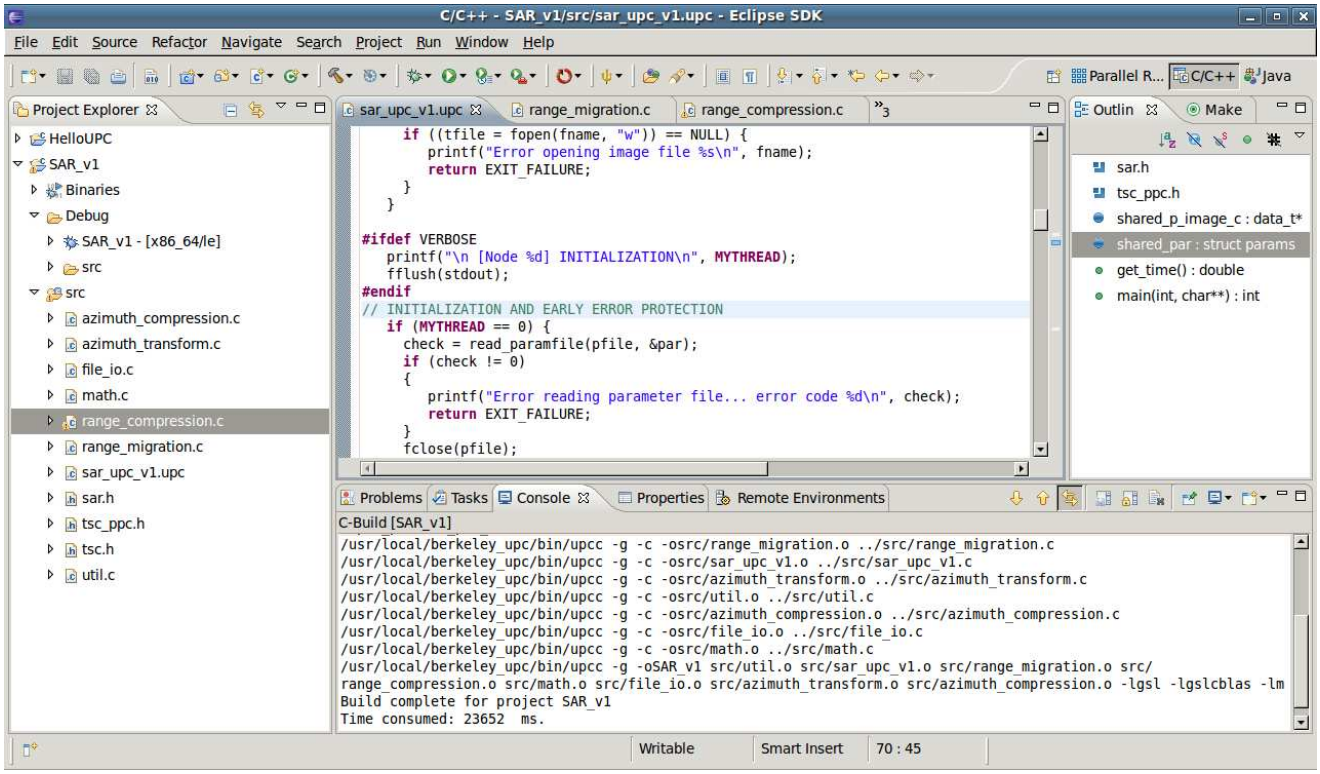

Figure 10. Eclipse workspace with SAR application.

we covered existing support for the UPC language before presenting UPC compiler toolchains—in particular our contributed support for Berkeley UPC—a UPC project wizard, and UPC analysis and user-assistance tools. We then focused on the integration of our PPW performance tool, which serves as a key example of the use of Eclipse as a PGAS tool integration platform, before finally describing a brief UPC application study as a concrete example of the process of UPC development in the Eclipse environment.

While Eclipse is presently a mature and state-of-the-art integrated development environment with useful features already available for the UPC programmer, the infrastructure provided by Eclipse, CDT, and PTP can be extended for more tools to specifically meet UPC development requirements. Furthermore, the wider PGAS community could benefit from future research and development efforts to bring more tools into a common, integrated environment like Eclipse. It deserves mentioning that another PGAS language, X10, also has Eclipse tooling available in the form of the X10DT. Besides the normal Eclipse IDE features of editor, build and launch features, X10DT also includes several refactoring tools that specifically enhance concurrency.

As we have briefly mentioned, one important area of future work involves debugging of UPC (and other PGAS) applications. One near-term step would entail integrating support for debugging of Berkeley UPC programs via GDB (using the capabilities described in [19]), even though this toolset is not a true UPC-aware debugger. A full-featured, open-source, UPC-aware debugger may likely become available in the future; when available, the full integration of this tool with PTP's parallel-debug framework should be considered essential. One other distinct opportunity for future work involves the remote enablement (via RDT) of the Berkeley UPC tool-chain and the PPW integration, though the latter would require PTP's External Tools Framework to first be extended to support remote operation.

In summary, the existing set of UPC tools exposes the opportunity for more tools to be integrated to aid the productivity of UPC programmers. As the UPC community becomes increasingly aware of the importance of tools—and recognizes the advantages of fully integrated toolsets—Eclipse will provide an appropriate environment for making UPC/PGAS development tooling a reality.

Acknowledgments

Some of the work presented in this paper was supported in part by the U.S. Department of Defense. Portions of this material are supported by or based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under its Agreement No. HR0011-07-9-0002. We would like to acknowledge former and current contributers to the CDT and PTP projects, including those at IBM Corporation and other supporting organizations. We would also like to express our thanks for the use of the SAR application written

by Hung-Hsun Su, former member of the UPC group at the University of Florida.

References

[1] *MPI: A Message Passing Interface Standard*, http://www.mpiforum.org, June 1995.

[2] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald, *Parallel Programming in OpenMP*, Morgan Kaufmann, 2000.

[3] The UPC Consortium, *UPC Language Specification v1.2*, http://www.gwu.edu/~upc/docs/upc_specs_1.2.pdf, 2005.

[4] K. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. Hilfinger, S. Graham, D. Gay, P. Colella, and A. Aiken, *Titanium: A High-Performance Java Dialect*, Concurrency: Practice and Experience, vol. 10, pp. 825836, 1998.

[5] *The X10 Programming Language*, http://x10-lang.org.

[6] *The Chapel Parallel Programming Language*, http://chapel.cray.com.

[7] *HPCS - High Productivity Computer Systems*, http://www.highproductivity.org.

[8] K. Kennedy, C. Koelbel, and R. Schreiber, *Defining and Measuring the Productivity of Programming Languages*, International Journal of High-Performance Computing Applications, vol. 18, pp. 441-448, 2004.

[9] C. Danis, J. Thomas, J. Richards, J. Brezin, C. Swart, C. Halverson, R. Bellamy, and P. Malkin, *Towards Applying Complexity Metrics to Measure Programmer Productivity in High Performance Computing*, Proceedings of the 2008 International Conference on Software Engineering (ICSE), Workshop on SE for Computational Science and Engineering, Leipzig, Germany, May 13, 2008.

[10] *Eclipse - An Open Development Platform*, http://www.eclipse.org.

[11] *The Eclipse Foundation*, http://www.eclipse.org/org.

[12] *Eclipse C/C++ Development Tooling*, http://www.eclipse.org/cdt.

[13] *PTP - Eclipse Parallel Tools Platform*, http://www.eclipse.org/ptp.

[14] G. Watson, C. Rasmussen, and B. Tibbitts, *An Integrated Approach to Improving the Parallel Application Development Process*, Proceedings of the 2009 IEEE international Symposium on Parallel & Distributed Processing, May 23 - 29, 2009.

[15] *Berkeley Unified Parallel C (UPC) Project*, http://upc.lbl.gov.

[16] H. Su, M. Billingsley, and A.D. George, *Parallel Performance Wizard: A Performance System for the Analysis of Partitioned Global-Address-Space Applications*, International Journal of High-Performance Computing Applications, accepted and in press.

[17] A. Jacobs, G. Cieslewski, C. Reardon, and A.D. George, *Multiparadigm Computing for Space-Based Synthetic Aperture Radar*, Proc. of 2008 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), Las Vegas, NV, July 14-17, 2008.

[18] *IBM xlUPC Compiler*, http://www.alphaworks.ibm.com/tech/upccompiler

[19] *Debugging Berkeley UPC applications*, http://upc.lbl.gov/docs/user/upc-debugging.shtml