Toward a Cloud IDE for HPC

Jeffrey L. Overbey and Mitchell C. Price

Department of Computer Science and Software Engineering
Auburn University
Auburn, AL 36849
{joverbey,mcp0038}@auburn.edu

Abstract

Eclipse Che is a relatively recent Eclipse Foundation project that provides a platform for building integrated development environments (IDEs) that run in a Web browser. This differs significantly from the Eclipse IDE as developers know it today, which is a traditional desktop application. The Eclipse Parallel Tools Platform has made remarkable progress in tailoring the (desktop) Eclipse IDE to the needs of scientific programmers. In this paper, we discuss the feasibility of moving PTP to the Web—building on Eclipse Che—and discuss how this approach may mitigate some of the challenges that currently face PTP.

1. Introduction

Many software engineers spend the majority of their time working in an integrated development environment (IDE) like Microsoft Visual Studio, Apple Xcode, IntelliJ IDEA, or Eclipse. Superficially, IDEs appear to be a convenience: they allow a programmer to write code, compile, debug, test, and perform version control tasks—all inside the IDE, using a user-friendly graphical interface. However, for large application development, IDEs are often more than a convenience: they are a necessity. Modern IDEs provide facilities for language-aware code search and navigation, static analysis, and refactoring that allow developers to understand and maintain large code bases efficiently.

Over the past few decades, IDEs and other software engineering tools have progressed substantially. However, software engineering *practices* have progressed as well (e.g., agile development, automated testing, and the use of design patterns). This continual advancement of tools and practices has allowed software engineers to develop reliable software at an ever-larger scale.

Scientists working in computational fields—such as computational chemistry and the geosciences—are developing increasingly complex computational models, many of which are designed to run on supercomputers that are also becoming increasingly complex. The increasing complexity of their software is causing these scientists to face problems well known to software engineers. However, most scientists have little or no training in software engineering, and they use languages and APIs (e.g., Fortran and OpenMP) that are generally not used by software engineers. So while scientific programmers could benefit from the tools and practices that are commonplace among software engineers, they cannot, due to differences in their working environments.

Eclipse is one of the most commonly used IDEs among Java programmers. One of its advantages is that it supports *plug-ins,* which are installed to add support for new programming languages or to add other features. The *Parallel Tools Platform*, or *PTP*, is a set of Eclipse plug-ins supporting HPC application development. PTP allows users to edit source code on remote HPC resources, compile using the remote system's compilers, submit and monitor batch jobs, and debug parallel jobs on the remote system [Alameda et al. 2012]. *Photran* provides Fortran language support (see Figure 1). Other Eclipse plug-ins can be installed alongside PTP and Photran to provide integration with version control systems (e.g., Subversion and Git), bug tracking systems (e.g., Bugzilla and Jira), and other tools. When Photran, PTP, and other plug-ins are installed into Eclipse, it can provide a solid foundation for HPC application development. However, there remain significant deficiencies and obstacles to its widespread adoption.

The Eclipse IDE, as developers know it today, is a rich-GUI desktop application. Eclipse Che is a relatively new project that provides a platform for building IDEs that run in a Web browser, rather than as desktop applications. IDEs based on Che are hosted on a Web server; developers write code, run, debug, and perform version control tasks all within their Web browser.

In this paper, we discuss the feasibility of moving PTP to the Web—building on Eclipse Che—and discuss how this approach may mitigate some of the challenges that currently face PTP. The paper is organized as follows. §2 describes some of the characteristics that make scientific software development unique. §3 describes PTP and its capabilities. §4 describes some of the challenges of developing an IDE for HPC—challenges PTP has faced. §5 describes Eclipse Che. Finally, §6 discusses the possibility of moving the capabilities of PTP to the Web, analyzing how a Web-based IDE could address some of the challenges faced by PTP.

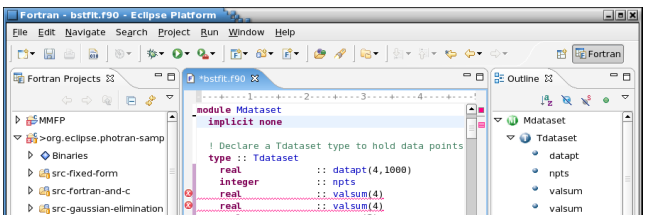

Figure 1. Editing a Fortran source file in Eclipse. The Projects view (left) shows the files in the active project; the Outline view (right) shows the high-level components—modules, subprograms, etc.—in the file currently being edited.

DEHPC'15 October 19, 2015, San Francisco, CA, USA.

2. Challenges of Scientific Software Development

Software developed by scientists—e.g., physicists, computational chemists, geoscientists, etc.—is fundamentally different from software developed by software engineers: Scientists use different languages and APIs, they target different hardware, and their software is developed with priorities and goals that differ from those of a software engineer. While scientists face many of the same challenges as software engineers, there are several factors that make their situation unique. Some examples follow.

Unique languages and APIs. Most scientific applications are written in either C/C++ or Fortran, with Fortran being the dominant language [Stitt and Robinson 2008, Loh 2010]. In both C/C++ and Fortran, MPI and OpenMP are typically used for parallelism. Python is increasingly popular among scientific programmers; MATLAB and R are prominent as well. While C, C++, and Python are used by many software engineers—nearly every student of computer science will encounter them—the same cannot be said of Fortran, MPI, OpenMP, MATLAB, and R; these remain more in the domain of the computational sciences.

Legacy and Community Codes. Many of the largest, most complex scientific models—including million-line earth system models key to understanding climate change—are written in Fortran. Many such scientific codes are "community codes," where many scientists in the same field of study rely on a single codebase for their work, modifying it as necessary to support their research. Rewriting these in another language is not realistic, at least not in the short term, due in part to the widespread intellectual investment in these codebases. Similarly, Fortran's 60-year history has left a wealth of legacy code that cannot be rewritten easily or cheaply. Legacy code becomes outdated over time: even if it works flawlessly, language constructs and programming practices become outdated but remain in code.

Lack of Software Engineering Training and Discipline. Of the scientists surveyed by Stitt and Robinson [Stitt and Robinson 2008], 90% were self-taught in programming. Most scientists have no formal training in software engineering, and many are unaware of software engineering best practices [Carver et al. 2013b]. Thus, scientific code tends to suffer from poor coding practices, a lack of unit tests, and even basic version control.

The Goal: Science. Scientists' ultimate goal is to produce science, not software; they are unlikely to adopt practices that they feel may impede their scientific productivity [Carver et al. 2013b]. (Anecdotally, automated testing is an example of such a practice.) Tools or practices that improve software quality will only succeed if scientists are convinced that they will improve the quality of the science they facilitate, not just the software.

Changing Target Architectures. Perhaps the biggest challenge to scientific programmers is that large scientific simulations are run on supercomputers. The lifetime of a supercomputer is about five years; thus, code targeted for these machines must continually adapt to changing architectures. Recently, the biggest architectural change is the incorporation of GPUs and similar accelerator devices; to take advantage of them, applications' compute kernels must be rewritten to use a language or API designed specifically for accelerator programming, such as CUDA, OpenACC, OpenMP 4, or OpenCL. The forthcoming move to exascale is anticipated to be even more disruptive.

3. The Eclipse Parallel Tools Platform (PTP)

PTP augments Eclipse with features necessary for HPC application development. Some of its most prominent features are:

Support for synchronized projects with remote compilation. A *synchronized project* contains source code that is saved on both the Eclipse user's laptop and a remote system. When the local code is modified, the files on the remote system are updated

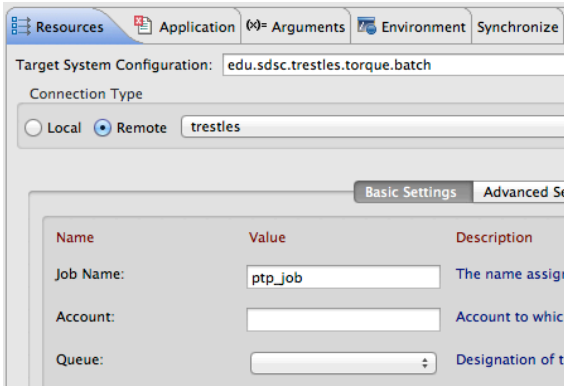

Figure 2. Submitting a batch job using PTP. The user can fill out fields in a dialog box rather than writing a job script.

within a few seconds, and when remote files are modified, the changes are copied back locally. This allows Eclipse's search, navigation, and refactoring features to operate on local code. However, when the user is ready to compile the code, it is done on the remote system (via SSH); errors from the remote compiler are identified and hyperlinked to locations in the local source code.

Job submission and monitoring. Once a project is compiled, it can be submitted to a remote job scheduling system. Job submission is done using a GUI; the user does not have to write a job script (Figure 2). The submitted job can be monitored and its output retrieved, all from within Eclipse. PTP supports TORQUE/PBS, Grid Engine, and SLURM, among others.

Graphical parallel debugger. PTP's debugger provides a graphical interface for debugging MPI and OpenMP programs on clusters. It provides the ability to monitor values and set breakpoints in user-defined process sets as well as in individual MPI processes and OpenMP threads.

Static analysis and code completion for MPI, OpenMP, and OpenACC. PTP includes code templates and library/API documentation for these APIs. In addition, it supports several static analyses, such as detecting mismatched MPI barriers.

An integrated SSH Terminal. Synchronized projects, remote build, job submission, and debugging are all supported by an SSH connection to the remote HPC resource. PTP also includes an integrated SSH terminal, so users can access a shell on the remote machine without leaving Eclipse. The SSH terminal is tightly integrated with Eclipse: for example, it is possible to launch an Eclipse editor in response to a command typed in the terminal.

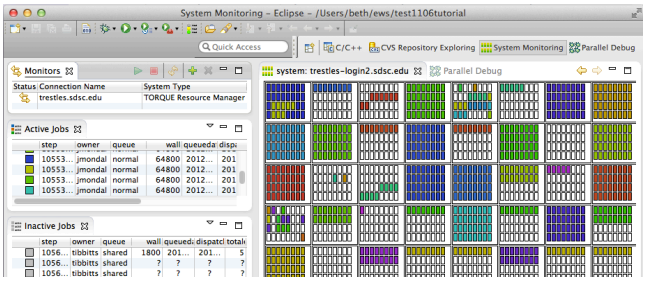

Figure 3. Job monitoring in PTP. Active and queued jobs are shown in the lists on the left; the view on the right shows active nodes, and the color-coded display indicates which jobs are running on which nodes.

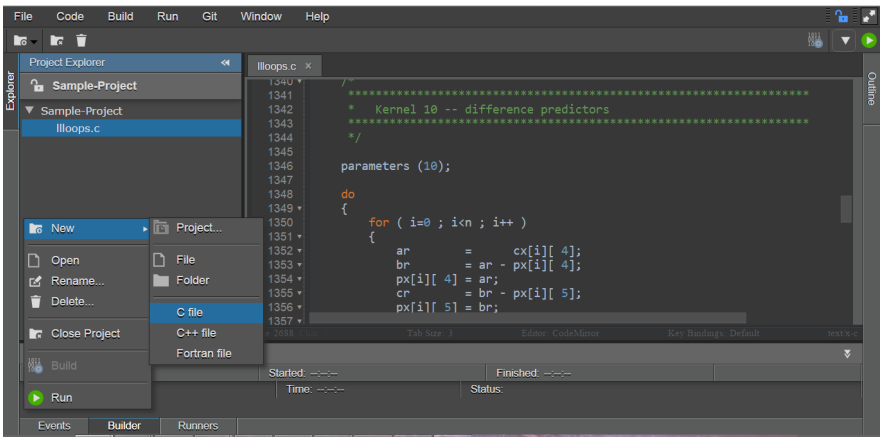

Figure 4. Eclipse Che. The user has opened a context menu (right-clicked) to create a new C source file.

The Eclipse C/C++ Development Tools (CDT) support C/C++ development in Eclipse. Photran supports Fortran development. PyDev supports Python development. When these plug-ins are installed together with PTP, Eclipse has nearly all of the functionality necessary for HPC application development.

4. Challenges of Developing an IDE for HPC

While PTP's capabilities are remarkable, it is not a panacea. It has not dominated HPC development in the same way that Visual Studio has dominated Windows development or Xcode has development OS X/iOS development. Most HPC developers do not use an IDE at all: they still use command-line tools. We have not conducted a formal study to understand this phenomenon, but there are several factors that we believe may play a role.

Scientists use what their peers use. Scientists' goal is to produce science, not software. By using the same tools as their peers, they have a guaranteed support network and the knowledge that their tools have worked for other scientists working on similar problems in the past. Using the same tools as their peers is the most obvious route to success; most scientists are unlikely to try new software development tools purely out of curiosity.

PTP doesn't "just work" out of the box. Most scientists considering PTP have a specific project they want to work on, and a specific supercomputer they intend to run that project on. Unfortunately, Eclipse and PTP require some configuration: the code needs to be imported into Eclipse, so the IDE can be aware of the entire code base; PTP must be configured so that it can establish an SSH connection to the remote machine and synchronize the project files on the local and remote machine. Unfortunately, this means that new users may be deterred from using Eclipse, since there is a nontrivial amount of up-front configuration necessary even to try it out.

The learning curve is steep. The Eclipse IDE is a complex piece of software. It has a steep learning curve even for software engineers familiar with other IDEs. Achieving fluency with the advanced features that make it exceptionally powerful (e.g., search, analysis, and refactoring) requires a time investment.

Technical support is limited. While there is a PTP mailing list, users do not have a help desk or technical support line they can contact for immediate support when they encounter a problem.

HPC vendors are not engaged. NVIDIA has several tools that are built on a customized version of Eclipse, but they cannot be installed alongside a "normal" Eclipse/PTP installation. The most popular parallel debuggers (DDT and TotalView) are not integrated with Eclipse. Cray builds standalone tools that are not integrated with Eclipse. With the exception of IBM and Intel, HPC vendors do not support Eclipse as an application development environment. This affects their users' decisions accordingly.

5. Eclipse Che

Eclipse Che (Figure 4) is an IDE that is hosted on a Web server and is run from the user's Web browser. In its current form, it is geared toward writing web applications. Che is designed to be "a kernel for loading, managing, and running extensions authored in Java that get translated into client-side JavaScript and server-side Java."[1] Thus, Eclipse Che is very modular and extensible, with an emphasis on allowing third parties to write their own plug-ins or extensions.

Che is a combination of a client-side editor and a server-side infrastructure that provides a set of microservices for the client to call. This architecture allows the client (e.g., the editor) to be interchangeable. Eclipse Che currently incorporates two different editors into its design: an open source editor written in JavaScript, known as CodeMirror, and an editor from Eclipse Orion, an Eclipse Foundation project that aims to create a browser-based tool integration platform for Web development.

Although its aspirations are broader, Che is currently targeted at Web application development. One of its key aims is to remove the hassle associated with workspace configuration. To this end, it builds a Docker image of the project and runs it in a Docker container. This gives developers much more control over the environment in which the project is executed, while also bypassing the process of ensuring a uniform development environment; this makes getting started on a new project much easier than with a conventional IDE.

Che is written in Java, with Maven as a build tool. It uses GWT to translate Java into JavaScript for client-side development, and it leverages the open source tool Everrest to help with server-side component creation. Eclipse Che is based on a code contribution to the Eclipse Foundation from Codenvy[2], a company that remains actively involved in its development. The Che project is in many ways still in its infancy; it will continue to grow to support new languages and platforms with increased functionality and effectiveness as the project matures.

[1] https://projects.eclipse.org/proposals/flare

[2] https://codenvy.com/

6. A Web-based IDE for HPC

Both PTP and Che are projects hosted by the Eclipse Foundation. Both are written in Java. Could the two have a synergistic relationship? If Che were augmented with PTP-like functionality for HPC application development, what would this look like, and would it be beneficial?

6.1 The Concept

As it exists now, Che allows users to create projects, compile them, run them, and debug them. Users can also perform version control using Git or Subversion. Projects are compiled, run, and debugged inside a Docker container running on the Web server.

In an HPC environment, projects would need to be compiled, run, and debugged on a remote HPC resource, not on the Web server. To be useful for HPC application development, many parts of PTP would need to be ported to Che. At a minimum, Che would need to support:

- *Remote connections and authentication.* While Che would be hosted on a Web server, it would need to be able to connect to a remote HPC resource to compile the user's project, submit jobs, access system header files, etc.
- *Synchronized projects.* The user's source code would need to be copied from the Web server to the remote HPC resource for compilation. PTP's synchronized projects provide this capability and more: they can also synchronize changes *back* from the remote HPC resource, and they allow filtering (e.g., there may be no reason to copy a documentation folder to the remote resource).
- *Environment modules and remote build.* Running the compiler on the remote HPC resource could be as simple as running `make`. However, most HPC resources support environment modules, which allow users to dynamically switch between compilers (or different versions of the same compiler) using a command like `module load pgi/15.5`. PTP includes a GUI that allows the user to customize the module(s) that will be loaded during a remote build; since environment modules are ubiquitous in HPC, this functionality would need to be present in a Che-based IDE as well.
- *Job submission and monitoring.* Finally, the user would need to have the ability to submit batch jobs to the HPC resource, as well as to monitor the status of those jobs and eventually retrieve their output after they execute. Again, PTP has an elaborate infrastructure that allows this to be done entirely from a GUI (without the need to hand-write job scripts), and it supports a large number of job schedulers, including PBS/TORQUE, SLURM, Grid Engine, and others. When it works, this is far friendlier than submitting and monitoring jobs from the command line, making this functionality especially welcome in a Web-based IDE.

This set of features is the absolute minimum necessary to compile and run HPC programs. Of course, there are many other features that would eventually be requested. Among them are:

- *An SSH terminal.* Power users will inevitably want to interact directly with the HPC resource; it is convenient being able to do this from inside the IDE. Furthermore, if Che encounters problems interacting with the remote HPC resource, it will be beneficial to have an interactive terminal that allows the user to issue commands through the same SSH connection established by the Web server.
- *Support for parallel application debugging and performance tuning.* Again, PTP includes an MPI debugger and TAU integration. These components would be more challenging to port to Che, but they are important nevertheless.

6.2 Advantages

Moving IDE functionality into the cloud has a number of advantages, considering the unique needs of scientific developers. Some advantages follow.

Minimal startup time. A Web-based IDE does not require any software to be installed on the user's computer. The user simply browses to a Web site and authenticates; all of the IDE functionality is immediately available in the browser. This can be particularly appealing to new users.

Fewer opportunities for client-side problems. The desktop Eclipse IDE can be affected by the user's operating system, Java virtual machine, local compilers and debugger, SSH configuration, and network connection or firewall. For a novice user, even getting "Hello, world" to compile in C/C++ can be a challenge. Many of these problems can be mitigated by moving most responsibilities to the server side, requiring little more than a Web browser and an HTTP connection from the user.

Minimal user interface. Due in part to its youth, Che has a much smaller feature set than the desktop Eclipse IDE. Since most scientists are self-taught in programming and unfamiliar with traditional IDEs, this minimalism is advantageous: the IDE provides only essential functionality, and it is presented in a less intimidating user interface.

Better opportunities for user study. PTP's developers are aware of the number of downloads, but they do not know the exact number of users, they do not know what languages or features are most commonly used, and they do not know what errors are most commonly encountered. When all users access a centralized server, these questions are far easier to answer. Likewise, the platform could be used to provide answers to many empirical questions about how scientists develop software.

Immediate updates and centralized error logging. Since most errors would (presumably) occur on the server side, they would be logged on the Web server; this would eliminate the need for users to individually report bugs. Bug fixes could also be deployed immediately on the server, rather than disseminating patches individually to all affected users.

A Che-based IDE for HPC could be particularly beneficial if a single provider (conceptually, a single Web server) provided the IDE to a large number of users. With appropriate modifications, the IDE could allow users to collaborate and share knowledge in ways that are not currently feasible in the desktop Eclipse IDE, where each user's installation is independent. Some potential scenarios follow.

Sharing of community codes. If Che were hosted on a central Web server, it would be easy for users to share projects. For example, one user could import and configure a community code (the Weather Research and Forecasting model, for example), then make it available for other users to copy and customize.

Sharing of bookmarks and Q&A. Taking the concept of collaboration a bit further, developers working on the same community code could share bookmarks—noteworthy locations within the source code—and the IDE could even provide a facility for users to ask questions about the code and have them answered by other users familiar with that codebase.

Collaboration and helpdesk access. If a user encountered a compilation error, the IDE could allow him to grant access to a collaborator (or helpdesk support engineer), who could log into the IDE, access his project, and see the error.

Sharing of HPC configurations. Compilation, job launching, and job monitoring vary from one supercomputer to the next. Sharing among users could also be beneficial here: one user—or a helpdesk engineer—could configure the IDE for a particular machine (e.g., by providing typical `qsub` and `qstat` commands),

then immediately make this available to other users with access to that machine.

In summary, the primary advantages a cloud IDE are twofold: (1) it could provide an environment that allows scientific developers to be productive quickly, with far less up-front configuration than the desktop Eclipse IDE; and (2) it could provide the ability for users to collaborate and share knowledge in ways that are not currently feasible with a desktop IDE.

6.3 Disadvantages

A cloud IDE has a number of disadvantages as well.

Cost. A cloud IDE cannot exist without a server, which in turn requires operations staff.[3] Even if the cloud IDE is free, open source software, there is necessarily some cost to operate it.

Loss of autonomy. The user must have an Internet connection to access the IDE, and all of the code is stored on the Web server rather than on the user's local machine. In short, a cloud IDE pushes a tremendous amount of responsibility to a central server, which has the potential to be a single point of failure.

Security issues are more complex. In a Che-based IDE oriented toward HPC, the IDE would be hosted on a Web server, but each user's code would still need to be compiled, run, and debugged on a different machine—an HPC resource. The Web server would need to act as a proxy, running commands on the HPC resource on the user's behalf. If the Web server were compromised, this could be disastrous. Protocols would need to be in place to minimize the amount of potential damage. On XSEDE resources, the recommendation is for the Web server to use OA4MP[4] to obtain a time-limited proxy certificate for the user, then authenticate with the HPC resource using GSI-SSH.[5]

Some features are difficult to provide in a cloud IDE. A good example is code completion: if the user types `foo.`, the IDE displays a list of methods that can be called on the `foo` object. Should the list of completions be computed on the client or the server? One cannot assume that there is a high-bandwidth connection between the user and the Web server, so computing completions on the server may not provide the responsiveness necessary to keep users happy. On the other hand, implementing this entirely on the client side (i.e., in JavaScript) is impossible, since it may require information from header files (for example) that are only available on the server.

6.4 Technical Considerations

We made an effort to begin prototyping some HPC capabilities for Che. While we believe this is feasible, there are a number of technical concerns.

The commercial Codenvy offering is quite mature (the Chrome Web Store offering of Codenvy claims over 89,000 users[6]). However, Eclipse Che—the open source version—is still new. At the time of writing, the code is still in the process of being migrated to the Eclipse Foundation, and the API is not finalized.

Second, Che plug-ins are substantially different from standard Eclipse/OSGi plug-ins. While many PTP components could be migrated to Che, a substantial amount of code would need to be written or rewritten. The functionality would need to be divided between the client and server, dependencies on some standard Eclipse plug-ins would need to be removed, user interface code would need to be rewritten for the Web browser, etc.

7. Conclusions

The concept of a Web-based IDE is not new. We have focused on Eclipse Che because it is derived from Codenvy, which is well known and commercially successful, and the possibilities for collaboration are great since both Che and PTP are Eclipse Foundation projects. Of course, a cloud IDE for HPC would not have to be built on this platform. Other Web-based IDEs include Cloud9[7], Codeanywhere[8], and Eclipse Orion[9], among many others[10]; some, including Cloud9 and Orion, are open source.

Kats *et al.* discuss many of the possibilities for Web-based IDEs and lay out a research agenda in this area [Kats et al. 2012]. In particular, they note areas in which Web-based IDEs need to mature. For example, most do not have an "offline mode," and integration with other development tools (continuous integration, version control, issue tracking, etc.) is limited. They also note how a Web-based IDE could be beneficial for programming education, and how it could provide synchronous real-time collaboration, somewhat like Google Docs, which could complement the asynchronous collaboration provided by version control systems.

Most existing cloud IDEs have focused on developing Web applications, often targeting platforms like Google App Engine. We believe that such an IDE could also be beneficial to scientific software developers, particularly those working in HPC. A Web-based IDE can provide a far friendlier user interface than the command line tools most developers currently use. Unlike a desktop IDE, it requires little from the client—only a Web browser (and an account on an appropriate HPC resource)—and it could provide opportunities for collaboration and knowledge sharing among users working on community codes. With careful design, a fusion of PTP and Che could produce a Web-based IDE offering many of the advantages of PTP in an environment that is amenable to the unique needs of scientific programmers.

References

J. Alameda, W. Spear, J. Overbey, K. Huck, G. Watson, and B. Tibbitts. 2012. The Eclipse Parallel Tools Platform: Toward an Integrated Development Environment for XSEDE Resources. In *Proc. XSEDE12*. ACM, New York, NY, 48:1–48:8. DOI: 10.1145/2335755.2335845

J. Carver, D. Heaton, L. Hochstein, and R. Bartlett. 2013. Self-Perceptions About Software Engineering: A Survey of Scientists and Engineers. *Computing in Science and Engineering* 15, 1 (2013), 7–11. DOI: 10.1109/MCSE.2013.12

L. Kats, R. Vogelij, K. Kalleberg, and E. Visser. 2012. Software Development Environments on the Web: A Research Agenda. In *Proc. Onward! 2012*. ACM, New York, NY, USA, 99–116. DOI: 10.1145/2384592.2384603

E. Loh. 2010. The Ideal HPC Programming Language. *Queue* 8, 6 (2010). DOI: 10.1145/1810226.1820518

T. Stitt and R. Robinson. 2008. A Survey on Training and Education Needs for Petascale Computing. Available from http://www.prace-project.eu/IMG/pdf/D3-3-1_document_final.pdf

[3] A user could self-host the cloud IDE on her laptop, but then it would have almost no advantages over the traditional Eclipse IDE.

[4] https://oa4mp.xsede.org/oauth/

[5] http://grid.ncsa.illinois.edu/ssh/

[6] https://chrome.google.com/webstore/detail/codenvy/lefigjbiimiemfhjmibbgemkpenelmag?hl=en

[7] https://c9.io/

[8] https://codeanywhere.com/

[9] https://orionhub.org

[10] See http://www.slant.co/topics/713/~what-is-the-best-cloud-ide for a list