



LAWRENCE  
LIVERMORE  
NATIONAL  
LABORATORY

**HIGH PERFORMANCE COMPUTING  
PRODUCTIVITY STUDY**

**EVALUATION OF PARALLEL TOOLS  
PLATFORM (PTP) ECLIPSE PLUG-IN**

*Lawrence E. Banks, Eveline I. Dube*

**June 29, 2012**

**Disclaimer**

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

## Contents

1	Introduction .....	3
2	Evaluation Hardware Platform .....	3
3	PTP Installation .....	4
3.1	Description .....	4
3.2	User Experience .....	4
3.2.1	Installation .....	5
3.3	Evaluation .....	6
4	Developing MPI Projects .....	6
4.1	Description .....	6
4.2	User Experience .....	7
4.2.1	ToolChain configuration .....	7
4.2.2	Out of the box MPI Example .....	7
4.2.3	Editing / Building .....	11
4.2.4	Resource Manager Configuration .....	14
4.2.5	Launching .....	17
4.3	Evaluation .....	19
5	Monitoring .....	20
5.1	Description .....	20
5.2	User Experience .....	20
5.3	Evaluation .....	20
6	Debugging .....	21
6.1	Description .....	21
6.2	User Experience .....	22
6.3	Evaluation .....	24
7	Evaluation Summary .....	26
8	References .....	28

## 1 Introduction

This report is a DARPA funded Work For Others (WFO) evaluation of the Parallel Tools Platform (PTP) version 5.0.3 (Released October 18, 2011). PTP is currently being developed and supported as an open source product managed by Project Lead Greg Watson and team at IBM plus several other contributors. The tool has been in development since 2005 with a goal of providing an infrastructure for the integration of high performance computing tools [1]. Basic functionality such as source code editing, compiling, launching, and debugging, plus more advanced features including performance monitoring and deployment have been bundled into an Eclipse Integrated Development Environment (IDE) plug-in to give parallel application programmers the features and improved productivity gained from using a common, well established development platform [2].

The evaluation encompasses the basic aspects of using PTP, beginning with the installation and configuration of the plug-in, followed by code development and debugging. Each aspect is evaluated based on the expectations of my experience as an Eclipse user and is presented in the following format:

- Feature description
- User Experience
- Evaluation

The feature description is based on the provided documentation for PTP and the ‘generic’ expectations of what the feature should provide. The user experience reveals the actual use and exercise of the feature – specifically in the context and perspective of the evaluation platform provided by Lawrence Livermore National Laboratory. The evaluation of each feature includes the overall outcome of the user experience, along with the expectations that were not met, met, or exceeded along with any issues that were encountered.

## 2 Evaluation Hardware Platform

The high performance computational infrastructure at Lawrence Livermore National Laboratory was utilized for this evaluation. At the time of this study, the specific hardware cluster used, named Sierra, consisted of the following architecture:

Sierra	
<b>Nodes</b>	
Login nodes: sierra[0,6,324,330,648,654,972,978,1296,1302,1620,1626]	12
Batch nodes	1,849
Debug nodes: sierra[12-27]	16
Total nodes	1,944
<b>CPUs (Intel Xeon 5660)</b>	
CPUs per node	12
Total CPUs	23,328
<b>CPU speed (GHz)</b>	2.8
<b>Theoretical system peak performance (TFLOP/s)</b>	261.3
<b>Memory</b>	

Memory per node (GB)	24
Total memory (GB)	46,656
Memory addressing	64-bit
<b>Operating system</b>	CHAOS 4.4
<b>High-speed interconnect</b>	InfiniBand QDR (QLogic)
Resource Manager	SLURM

**Table 1: Sierra cluster architecture specifics [3].**

PTP allows three basic configurations for the development environment. A *local* environment is where Eclipse and source code development resides on the target machine. A *remote* environment allows the developer to use Eclipse on a local machine while maintaining code development on a remote target machine. A third configuration is termed a *synchronized* environment which is a hybrid of the first two; Eclipse runs on a local machine with source code that resides, and is kept in sync, on both the local and remote target machines. This evaluation has been conducted exclusively with the *local* environment configuration – Eclipse with PTP and source code resides in a user environment on the Sierra cluster.

Several compilers are available on the Sierra cluster. This evaluation focused on parallel coding using the Message Passion Interface (MPI) libraries with an Intel compiler for the native chipset of the Sierra architecture.

## 3 PTP Installation

### 3.1 Description

Typically, an Eclipse plug-in will have an update site, i.e. url, that can be added within Eclipse via a software update configuration. Once Eclipse knows about an update site it can automate the download and install and updates of the plug-in. This process will identify any dependencies the plug-in may have and prompt the user to allow installation of these needed dependencies as well.

PTP employs the concept of a *resource manager proxy* to bridge communicating between the Eclipse environment and the resource manager of the target parallel environment. PTP supports several resource managers and requires the desired proxy be built from source on the target machine to ensure proper configuration.

PTP comes bundled with a parallel debugger with basic debugging capabilities of setting break points and stepping through program execution. As with the resource manager proxy, the debugger must be configured and built from source on the target machine.

### 3.2 User Experience

The Eclipse IDE is primarily written in java, thus a java virtual machine, or jvm, is required. The latest versions of PTP (and Eclipse) require java 1.5 or greater. Since PTP appears to be regularly updated with bug-fixes and additional features, it was decided to ensure a newer java 1.6.x version was available. Checking the java version for the Sierra environment:

## PTP v5.0.3 Evaluation

```
➤ java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) 64-Bit Server VM (build 1.6.0-b105, mixed mode)
```

This was not the newest version of Java, but was more than sufficient for the duration of this evaluation.

### 3.2.1 Installation

The PTP plug-in has matured to the point where it is included with a full Eclipse download, thus there are two ways to install the PTP plug-in: install a new full Eclipse bundle containing PTP, or install the PTP plug-in into an existing Eclipse environment. The method used in this evaluation was to simply install the full Eclipse bundle for parallel application development containing the PTP plug-in. This download was obtained from the Eclipse download site: <http://www.eclipse.org/downloads> by selecting the “Eclipse IDE for Parallel Application Developers” bundle. Version 5.0.1 of PTP was originally installed with this bundle. The update mechanism in Eclipse was used to update the PTP version to 5.0.3. For ease of use, and to allow for an independent Eclipse process, an alias was created for the initiation of Eclipse sessions:

```
➤ alias eclipse='nohup ~pathToEclipse/eclipse -showlocation &'
```

The configuration of the PTP plug-in was necessarily more involved than a basic Eclipse plug-in. This was mainly due to the need of ensuring the resource manager and parallel debugger were configured and built for the specific target parallel environment.

#### 3.2.1.1 Resource Manager Proxy

To communicate with the SLURM (Simple Linux Utility for Resource Management) resource manager, a resource manager *proxy* must be configured and built from source. The source code for the proxy is included in the download of the initial install, and any further update of PTP. The following steps were specific to the evaluation environment on the LLNL Sierra cluster and are based on the PTP installation documentation bundled with the plug-in and accessible via the standard Eclipse *Help* facility [4].

- 1) Ensure `slurm.h` and `slurm_errno.h` include files exist on the target system.
  - These files are located in `/usr/include/slurm`
- 2) Build PTP proxy and utils libraries which are included with the plug-in and are found relative to the plugins directory of the Eclipse install. Note: the subdirectories correspond to the current version of the plug-in installed. Therefore, this will need to be redone each time the plug-in is updated (i.e. new version installed).
  - a. `cd` to `org.eclipse.ptp.proxy_5.0.3.201110141146/`
  - b. NOTE: these directories were found to contain README and INSTALL files. Considering these are bundled with the code it was assumed that they would contain more current information and thus would override any conflicting documentation in the install guide.
  - c. Run `autoreconf`
  - d. Run `configure`
  - e. Run `make`
- 3) Build the PTP SLURM proxy.

- a. cd to org.eclipse.ptp.rm.slurm.proxy\_5.0.3.201110141146
- b. Run autoreconf
- c. Configure the makefiles corresponding to the SLURM on the system.  
Run configure --with-slurm=/usr/include/slurm

- d. Run make

NOTE: Instructions in INSTALL file say to run 'make install'. However, this attempts to place the ptp\_slurm\_proxy into the /usr/local/bin directory – which, of course, is a read-only directory for typical users. Later documentation indicates the ptp\_slurm\_proxy is to be found in this org.eclipse.ptp.rm.slurm.proxy.xxx directory, which is what make does by default, so it was assumed that 'make install' was not needed.

### 3.2.1.2 Parallel Debugger

This PTP release bundle contains the initial release of the PTP Scalable Debug Manager (SDM). The SDM consists of an MPI-based client/server framework that can attach to backend debuggers, a proxy library that allows control from a remote driver, and a Java JNI library to allow communication with the Eclipse interface.

- 1) Relative to the Eclipse/plugins install directory:
- 2) cd to org.eclipse.ptp.debug.sdm\_5.0.3.201110141146
- 3) Build the SDM debugger
  - Run configure
  - Run make

## 3.3 Evaluation

Documentation was quite good. PTP is unusable without proper setup for the specific target machine and knowing what pieces to build would be impossible without basic documentation. However, some documentation was inconsistent with the bundled README and INSTALL instructions. A higher-level install script that would configure and build all the necessary pieces on the target machine would be useful.

## 4 Developing MPI Projects

### 4.1 Description

A clean integration with Eclipse will successfully inherit a set of common development features including a managed build system, syntax-aware editing for supported languages, integration with external tools – specifically compiler and library configuration.

## 4.2 User Experience

### 4.2.1 ToolChain configuration

For each project created, a *toolchain* must be defined specifying the desired compiler, linker and associated libraries required for building the executable. PTP allows defaults to be defined in the Parallel Tools general settings (see Figure 1).

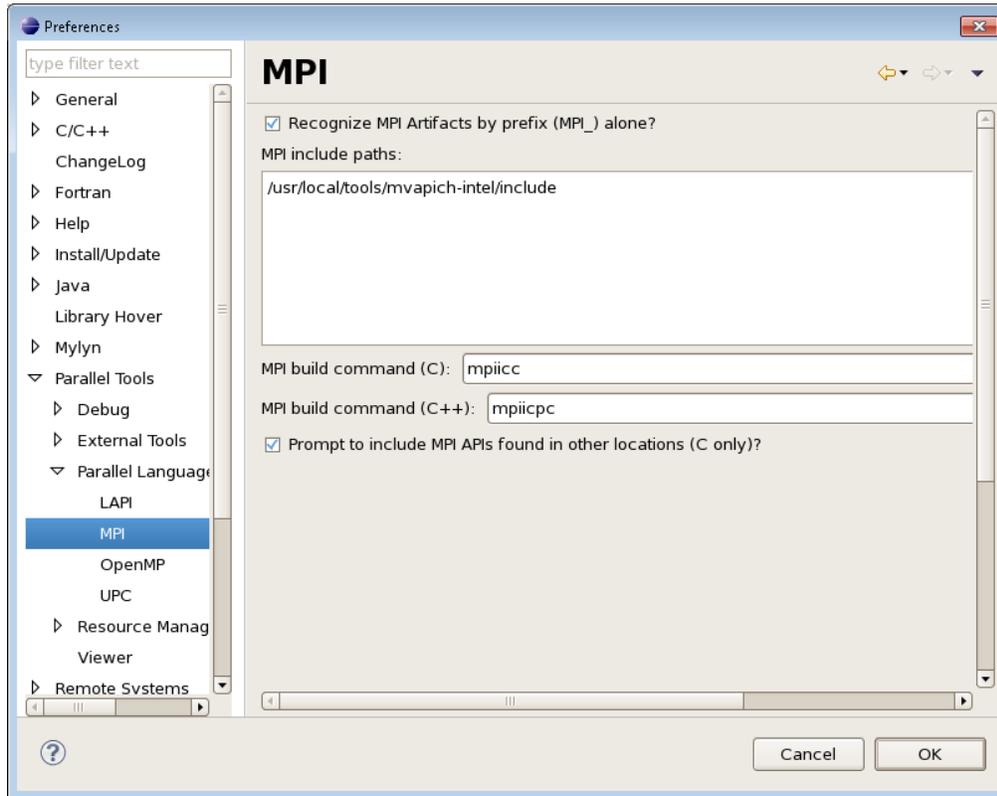


Figure 1: General settings for MPI.

Here the library path for MPI include files is defined, as well as the compiler command for the Intel C and C++ MPI compilers. These defaults can be used or overridden when a specific toolchain is defined for a project.

### 4.2.2 Out of the box MPI Example

Following the online help documentation the example MPI project was created (see Figure 2). Since this project is created from scratch it will be a managed build project – meaning that Eclipse manages the build using the compiler, linker, and libraries as configured by the user.

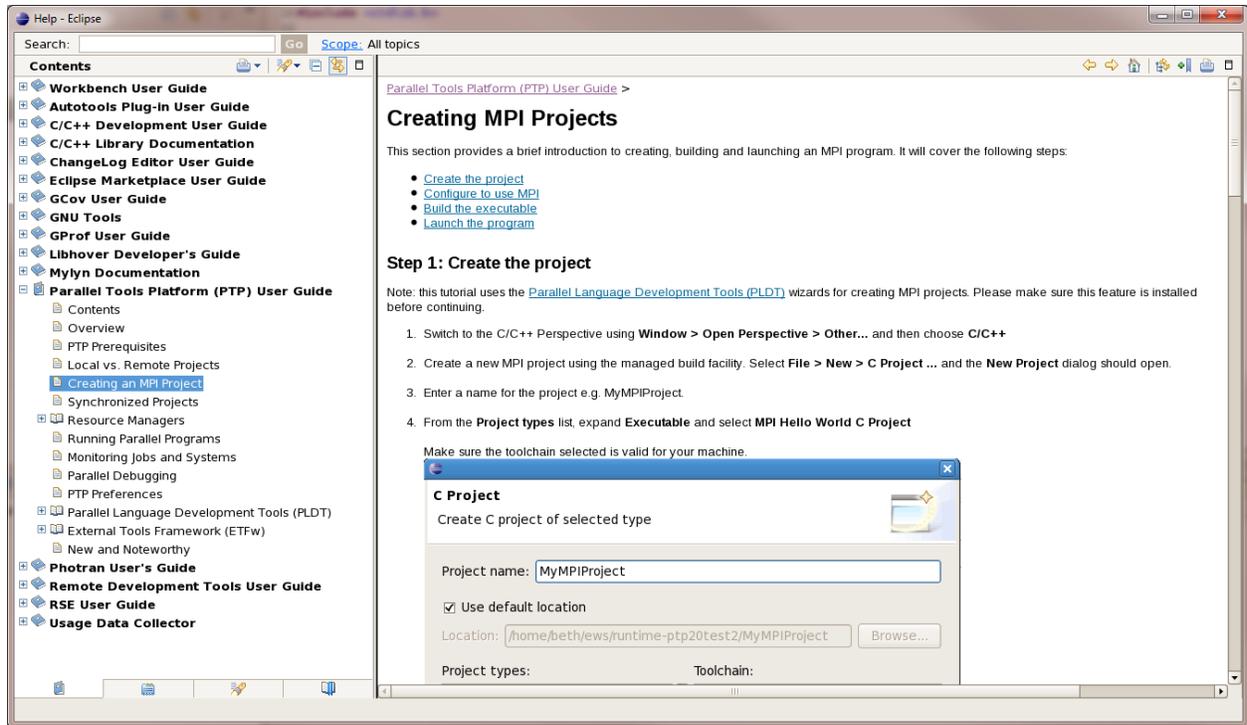


Figure 2: Example MPI Project via PTP online help.

The Eclipse new project wizard was used to name the project and select the toolchain (see Figure 3).

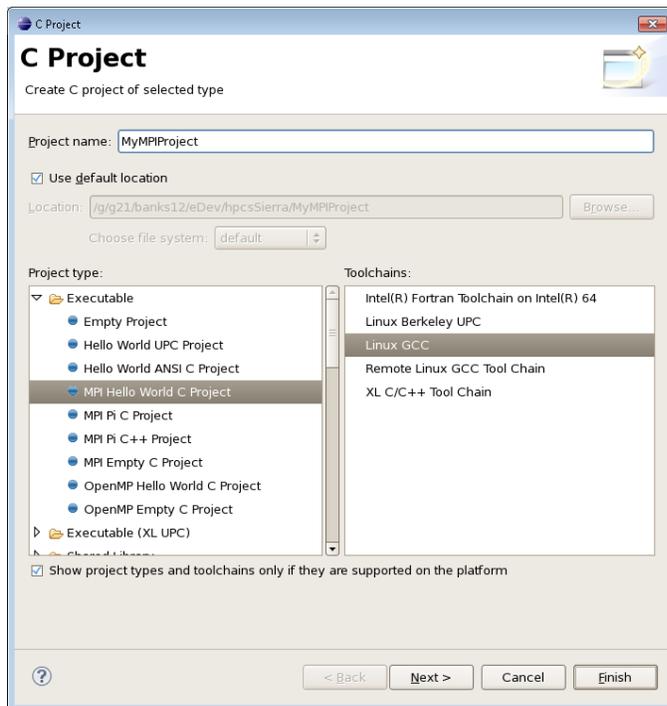


Figure 3: Eclipse New Project wizard.

Note how the Linux GCC toolchain was chosen. This was due to the fact that there was no appropriate toolchain available for selection for the Intel MPI compilers to be used. However, as will be shown later, it was possible to modify the Linux GCC toolchain to contain the libraries needed for the desired compilers.

The next wizard screen (Figure 4) contained the expected default settings as previously defined in the general settings for PTP – namely the include path and the compiler command. At this point, the toolchain could be tailored for specific projects.

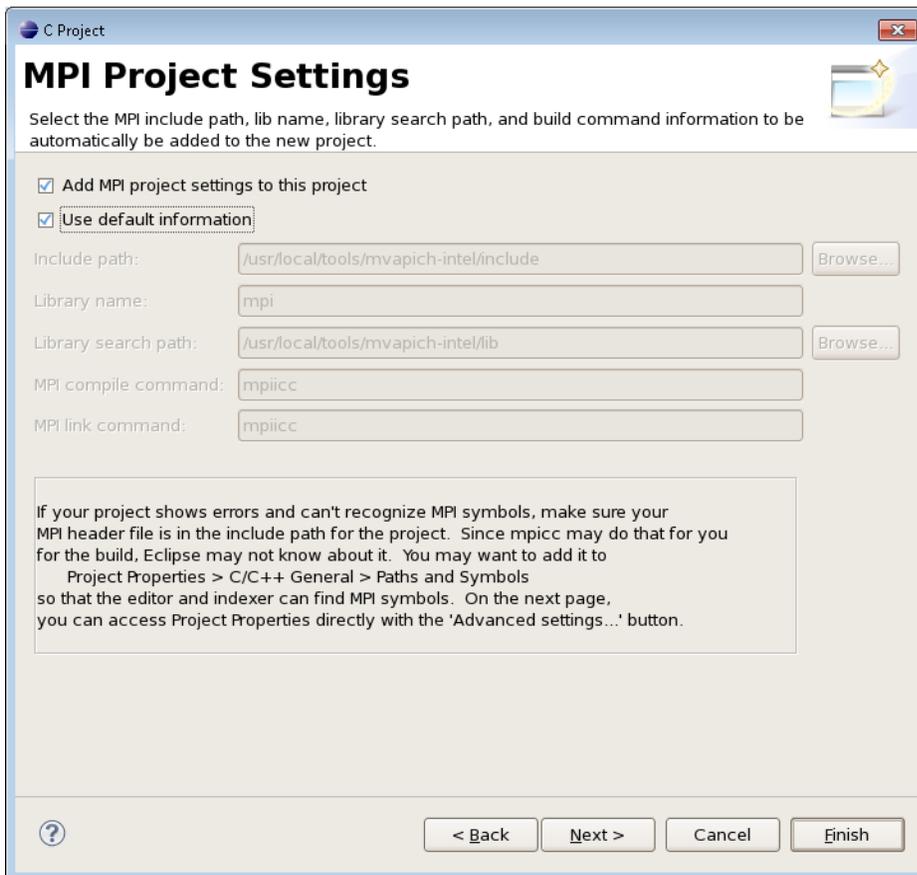


Figure 4: MPI default project settings.

The next screen shows configurations that can be selected for the project. Options include a debug configuration enabling a build to maintain source and variable references, and a release configuration which allows the compiling/linking phases to be optimized. There must be at least one valid configuration present. This is one reason the Linux GCC Toolchain was selected to be modified – it appeared to be the only C/C++ toolchain with valid configurations (see this section’s Evaluation).

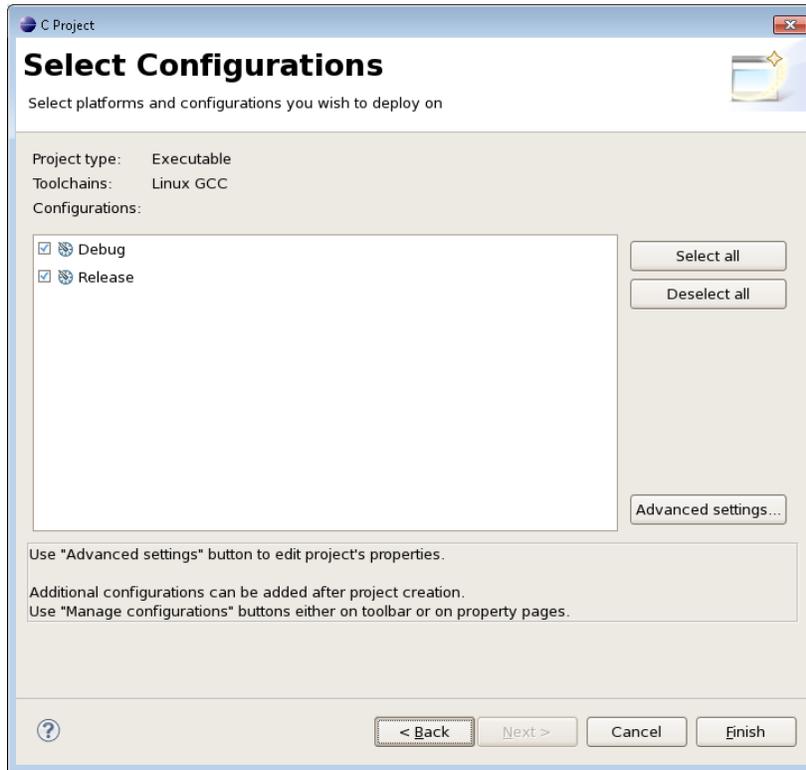


Figure 5: Toolchain configurations.

Finishing the wizard resulted in the creation of the example 'Hello World' program (Figure 6).

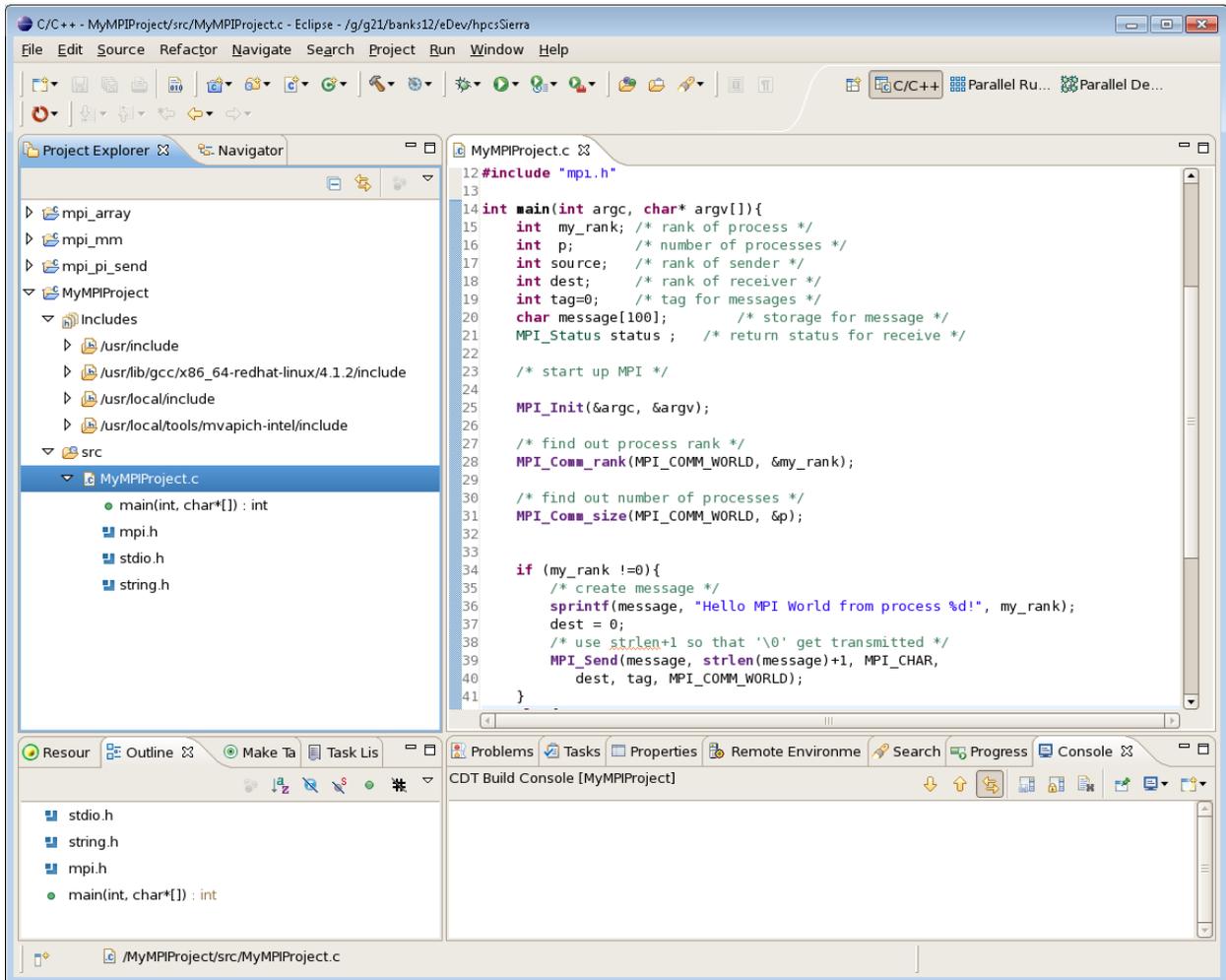


Figure 6: Out of the box example MPI C 'Hello World'.

### 4.2.3 Editing / Building

The PTP Toolkit is built upon Eclipse CDT (C/C++ Development Tooling) plugin module [5]. Thus the expected bells and whistles of integrated source code editing are present. This includes color coded syntax, global searches, hot links to function declarations and references, code completion and context sensitive help. The plugin also includes automatic building, incorporating compiler errors and warnings output to the user.

After creation of the example project, the next step is to attempt to compile and link the code to create an executable. The output of this effort is shown in console window (see Figure 7).

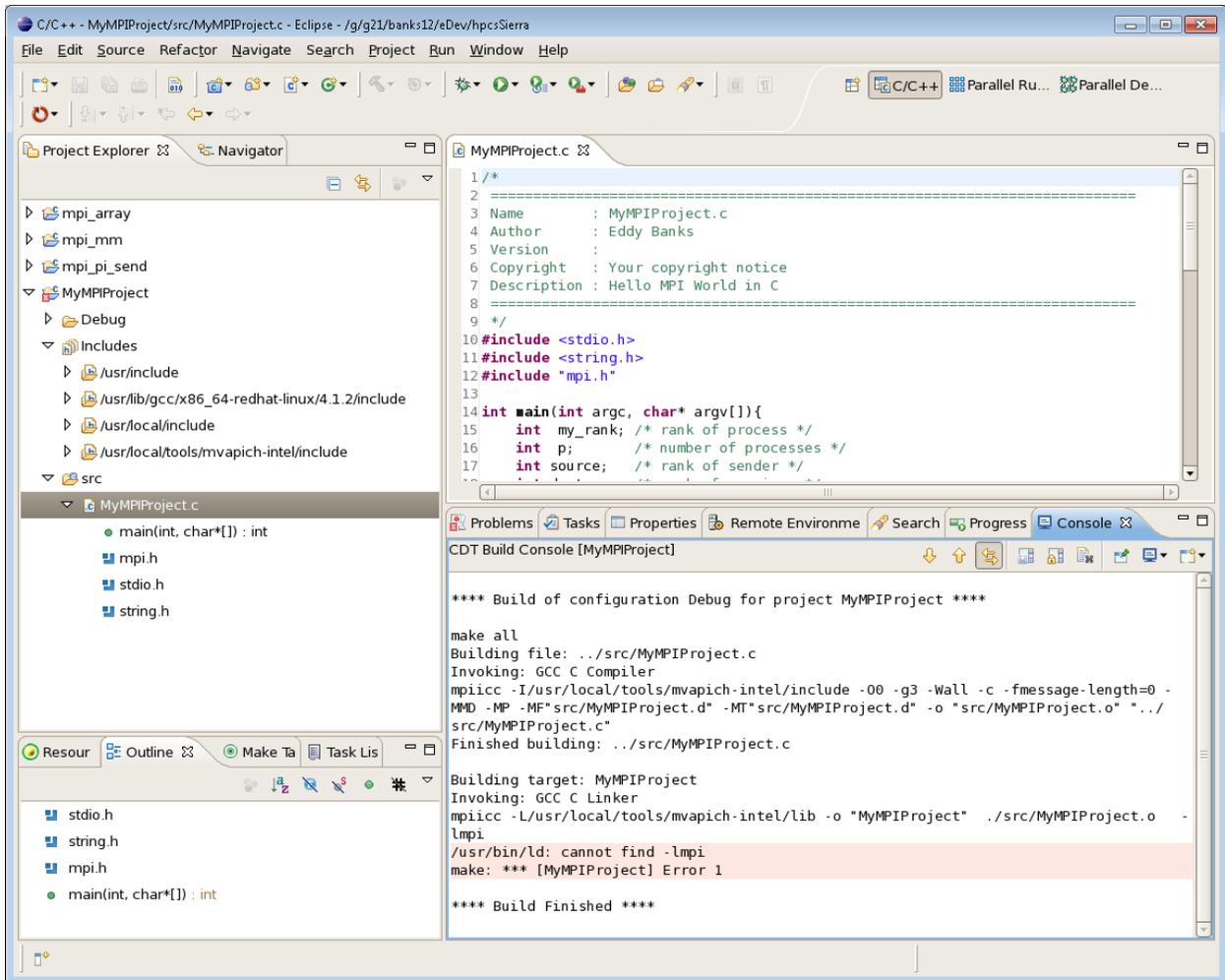


Figure 7: Output of build shown in Console view.

There are two items to note in the console window. One, although the output states that it is invoking the GCC compiler and linker – likely due to the fact the Linux GCC Toolchain was chosen as the one to be modified – it can be seen that it did correctly invoke the desired Intel compiler and linker as previously configured in the general setup. And two, the build failed due to the linker not being able to find the ‘mpi’ library. The build error was fixed by going into the project’s properties: C/C+ Build|Settings and removing the library search path ‘/usr/local/tools/mvapich-intel/lib’ from the linker search path, and ‘mpi’ from the linker libraries (see Figure 8).

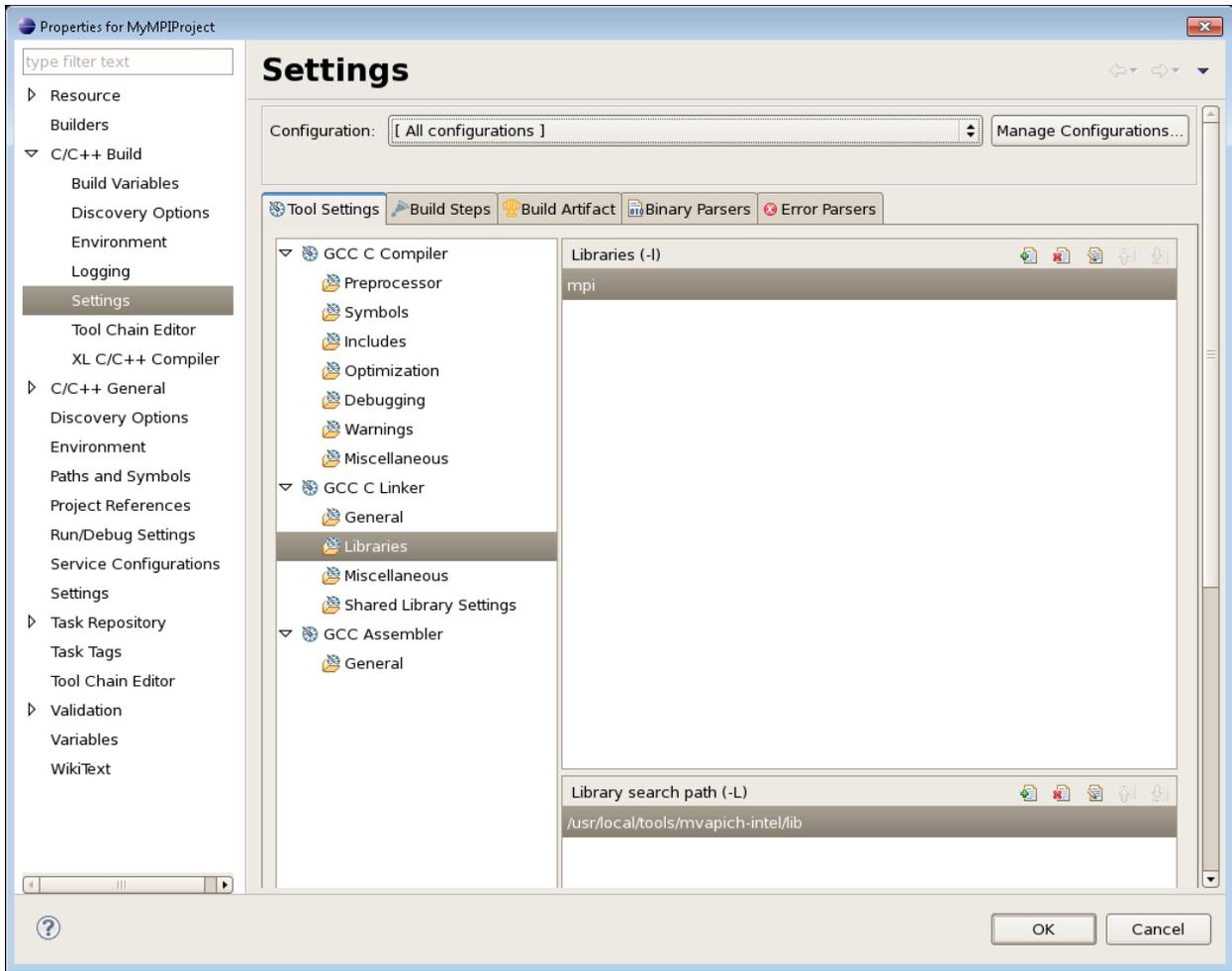


Figure 8: MyMPIProject Settings for C/C++ Build.

Now, reattempting the build results in success (see Figure 9), and an executable is created in the Binaries and Debug folder.

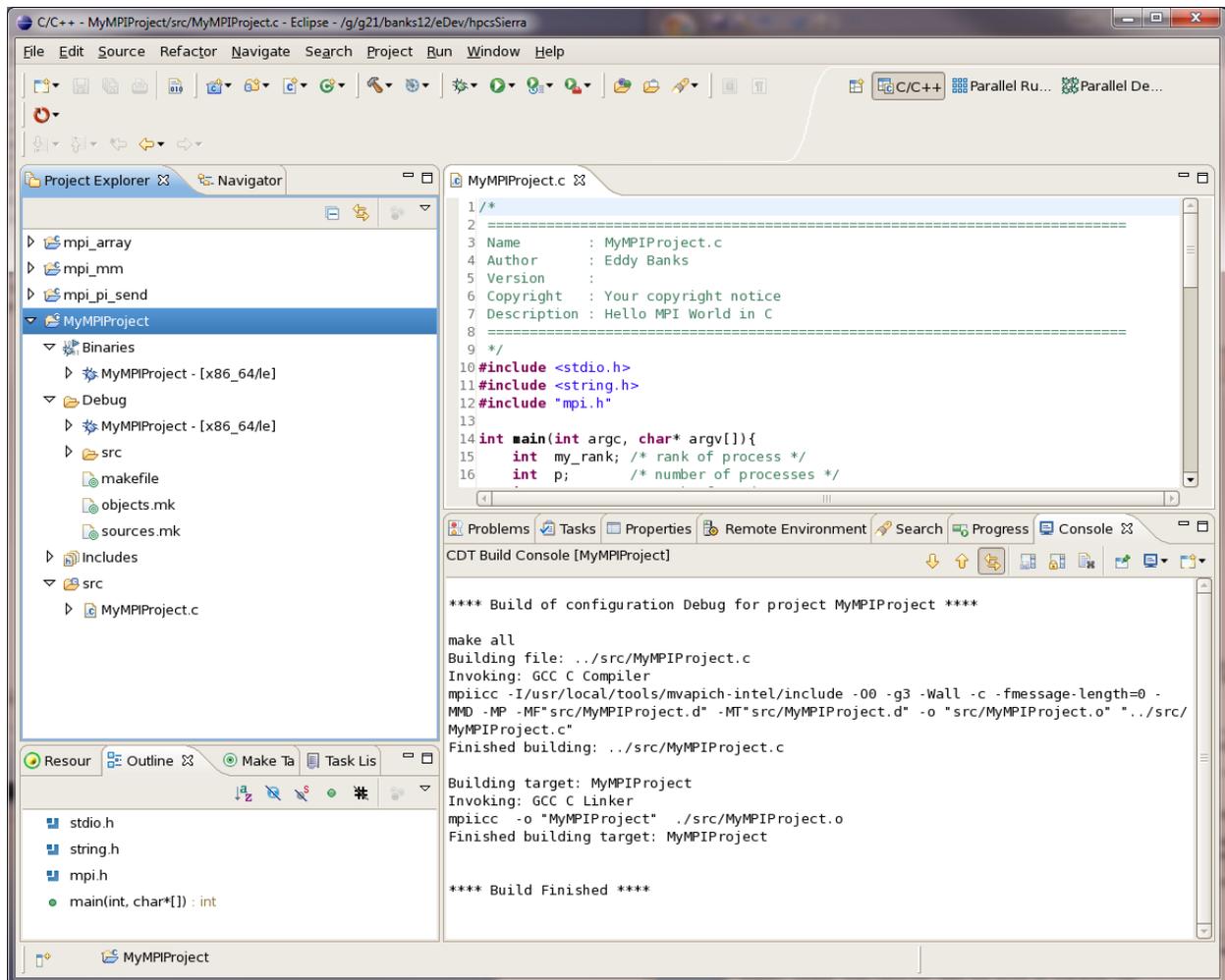


Figure 9: Clean build of MyMPIProject example.

#### 4.2.4 Resource Manager Configuration

Running an executable on a parallel system requires configuring and submitting a *job* to the system's resource manager. Recall earlier that a proxy to the resource manager was built from source as part of the installation of PTP. This proxy must be configured within Eclipse and must be available to coordinate the launch of an executable.

To configure the resource manager, PTP provides a *Parallel Runtime* perspective. A *perspective* in Eclipse is a top-level window that contains *views* for a particular application. As such, the Parallel Runtime view contains views for the resource manager(s), managed jobs, hardware cluster information, and output console (see Figure 10).

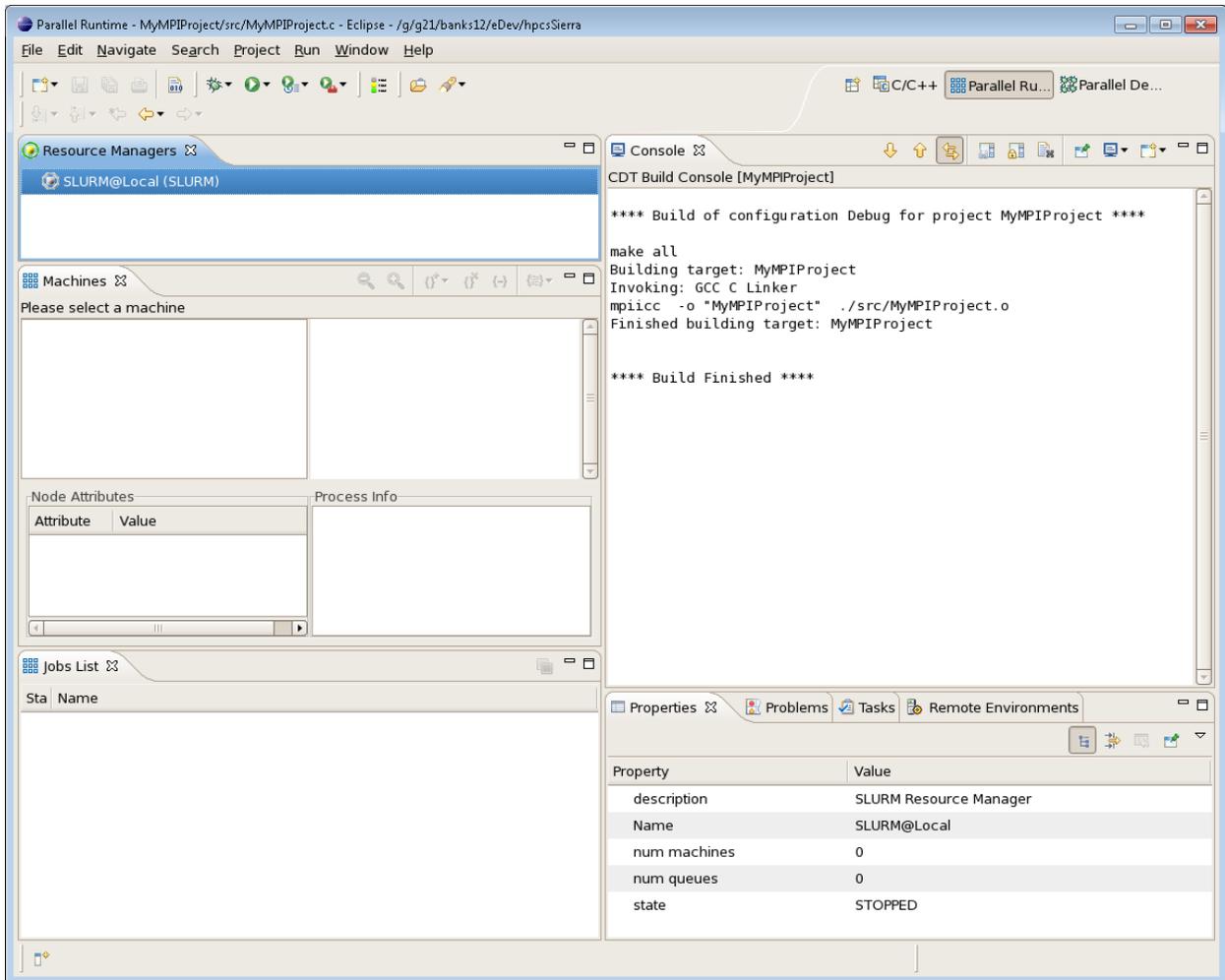


Figure 10: PTP Parallel Runtime perspective views.

From within the 'Resource Manager' view, right-click the mouse to see a dialog allowing to 'Add Resource Manager...'. This brings up a wizard to add and configure a resource manager (Figure 11).

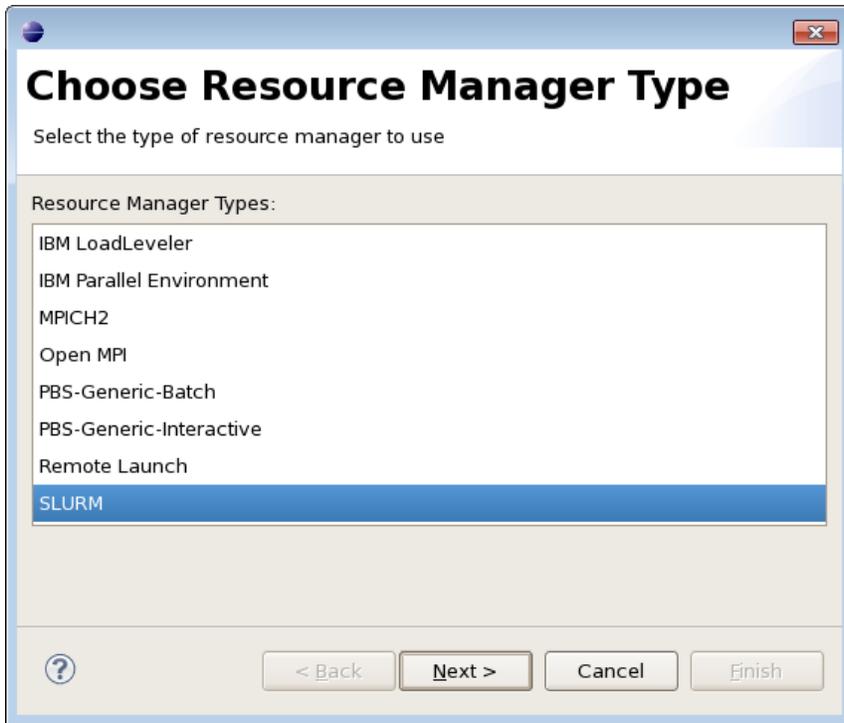


Figure 11: Add Resource Manager wizard.

Since SLURM is the resource manager on the Sierra system, SLURM is the chosen type. The wizard simply prompts for the location of the proxy for the resource manager which was built during installation, specifically: 'org.eclipse.ptp.rm.slurm.proxy\_5.0.3.201110141146/ptp\_slurm\_proxy'

The proxy is started from the 'Resource Managers' view. Right-click on the resource manager and choose 'Start Resource Manager'. With a running resource manager (see Figure 12) the views in the Parallel Runtime perspective show a color-coded status of the machine nodes: green for available, yellow for busy. Clicking on a node displays basic information about that node in the properties view.

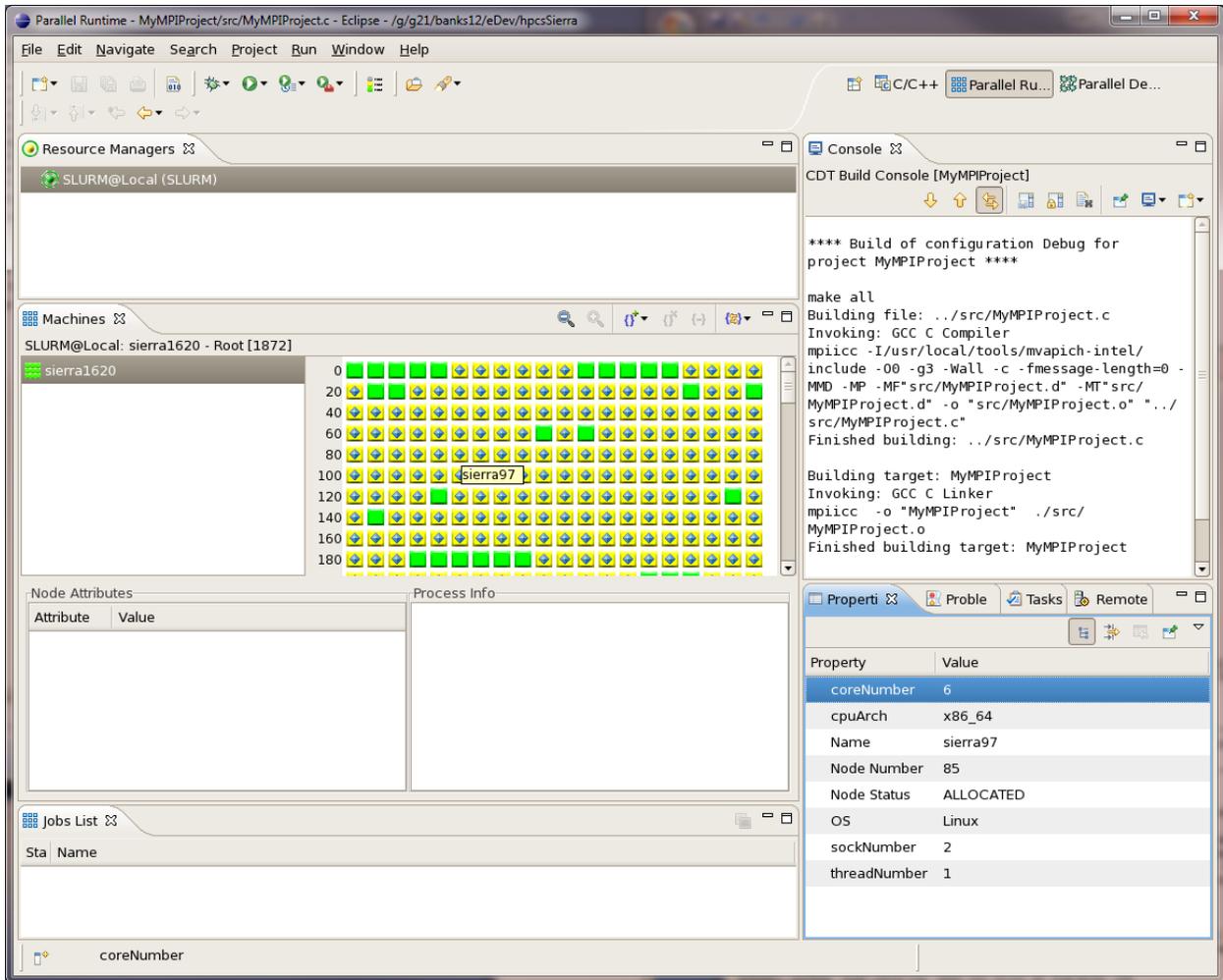


Figure 12: Parallel Runtime perspective showing machine and node information.

#### 4.2.5 Launching

With a cleanly built executable and an available proxy to the SLURM resource manager, a run configuration can be created. From the main bar choose Run|Run Configurations... (see Figure 13) .

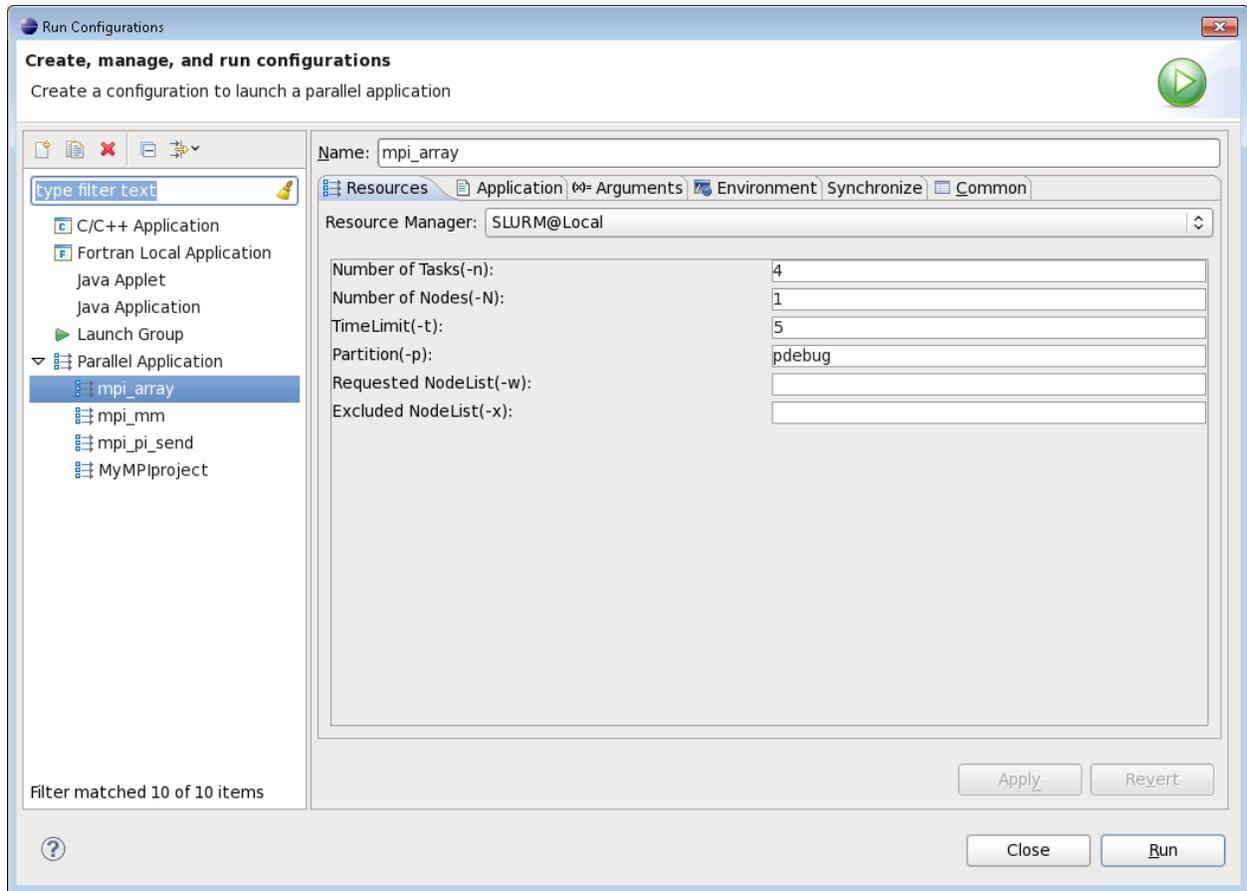


Figure 13: Run Configuration panel

For a configuration, the desired resource manager is assigned along with the number of tasks, nodes, time limit, and partition for job submission. If desired, one can request specific nodes as well as exclude specific nodes. The application to run is specified from the Application tab, and any arguments to be passed to the application may be specified via the Arguments tab. It is trivial to create several different run configurations for the same application.

To submit a run configuration, right-click on the pull-down arrow to the right of the Run icon (see Figure 14). The act of running a configuration will run make on the application to ensure the binary is current with respect to any source code modifications. Then the job is submitted to the SLURM manager for execution on the cluster. Any output generated from the running processes is reflected in the console view.

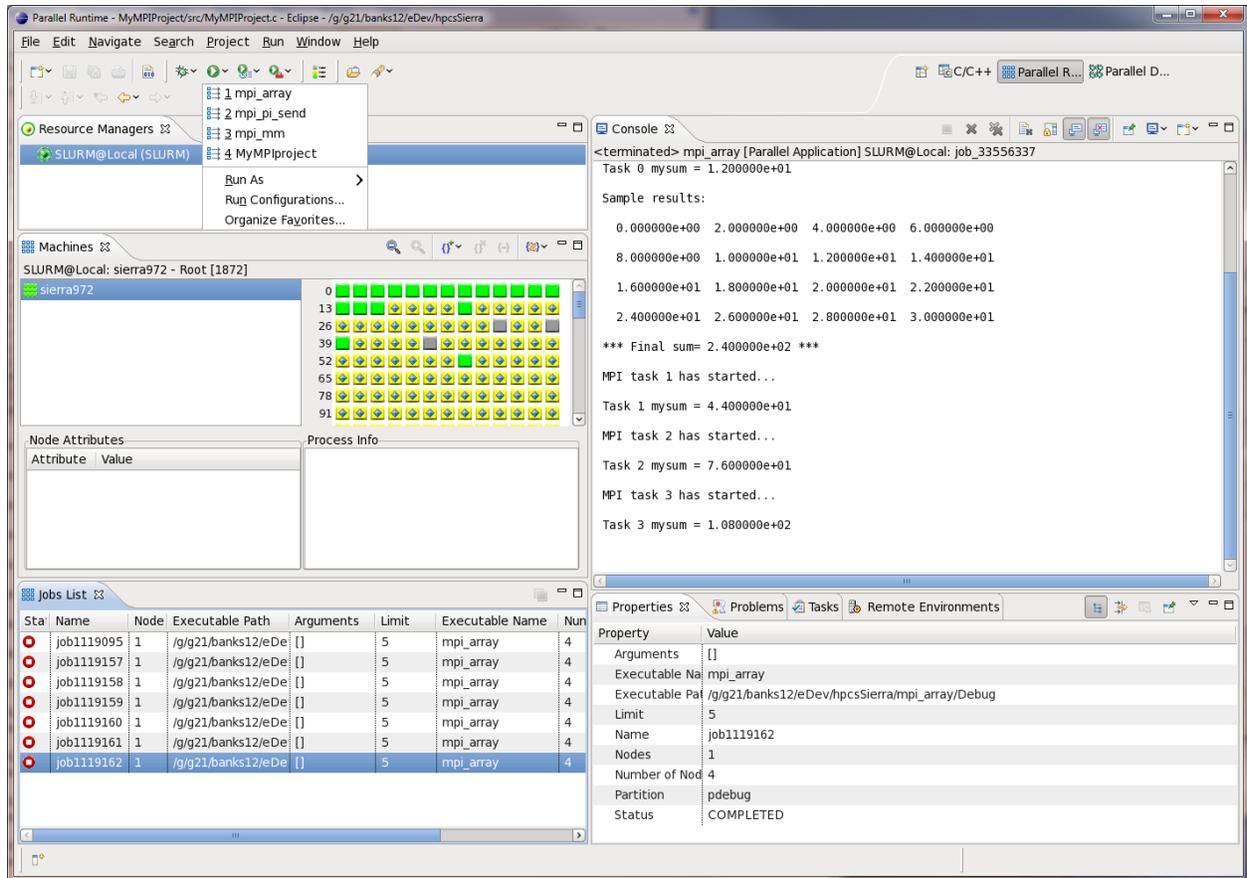


Figure 14: Running a job.

### 4.3 Evaluation

The PTP plugin adheres to the basic functionality of the Eclipse IDE in that it provides the framework for automatic building of a source code project. Wizards are provided to aid in the initial configuration of the compiler and linker. This type of configuration is typically called a toolchain. PTP includes a few predefined configurations for some common development environments including a basic Linux GCC toolchain. However, if the desired toolchain is not in the set of predefined configurations then one of the existing configurations must be modified to point to the desired compiler, linker, and libraries. Being forced to use an existing configuration to build the needed configuration can appear confusing. As was experienced previously, it was necessary to modify the 'Linux GCC' toolchain to create an 'Intel C++' compiler/linker setup. There should be an option to create and configure a user-defined toolchain from scratch. Additionally, it would be helpful to have the ability to copy a predefined configuration that could be renamed and then modified. However, even with this small issue, the configuration of a toolchain is a tremendous aid in that once configured it can be reused for multiple projects alleviating the need to build a new configuration each time.

The PTP has been integrated nicely with the CDT plugin which is designed for developing C/C++ projects. As such, all the expected features found in a code development Eclipse plugin are available. These

features include mechanisms for automated building, code editor with syntax highlighting and source completion, button-click navigation to function definitions and references, and source code refactoring in the context of the entire project. The *integrated* capabilities provided by this set of features have the greatest positive impact on a software developer's productivity.

Since the purpose of PTP is to aid in the development of parallel processes, it must address the added complexity of submitting the compiled code to the job management system of a target node cluster. PTP includes the concept of a resource manager proxy which acts as a go-between for the Eclipse development environment and a cluster's job management system. PTP allows the creation of run configurations that refer to a particular resource manager proxy to enable the execution and monitoring of the job from within the IDE. The resource manager proxy used in this evaluation was configured for a SLURM management system. For each run configuration, PTP exposes a subset of the more common parameters used by the SLURM system.

## 5 Monitoring

### 5.1 Description

Once a job is submitted to the resource manager of a cluster it's vital to be able to monitor its status. Jobs are placed in a queue and will compete with other jobs for node resources. A user will need to know what state a job is in (e.g. pending, running, completed), and have the capability to remove a job from the queue or cancel a running job.

### 5.2 User Experience

There are several capabilities available on the Parallel Run perspective for monitoring submitted applications. Referring back to Figure 14, a visual view of the nodes in the cluster is provided with icon status showing the state of each node as being idle, allocated or down.

The *Jobs List* view shows a table of information of the submitted jobs owned by the user consisting of the parameters provided in the run configuration along with the assigned job number and the current status of the job. By clicking on a job in this table the details are shown in the *Properties* view. Output generated from each job is displayed in a *Console* view. Each job's output is captured by a new console instance and can be specifically selected to view, or re-view, the output for that job. A pending or running job can be easily cancelled in the Jobs List view by clicking on the job and then clicking on the red square icon at the top of the view.

### 5.3 Evaluation

The Machine view is quite nice in that properties of any node can be explored and the load of the cluster as a whole can be observed. This view proved to be quite responsive as a specific node can be seen to change from available to allocated and back to available as an application configured for this node is executed.

The table view of submitted jobs easily aids in productivity in three ways. One, by simply keeping a list of the submitted jobs a user need not manually query the system for jobs in the queue. Two, by a simple

click on a job displays the submitted job's status and allows cancelling of the job if desired – again, saving manual queries and commands to the queue. And three, keeping track of the output of each completed job avoids in programmatically routing output to specific output files and provides quick review of the output of any completed or running job.

As a user establishes an iterative development procedure for editing/running/testing/debugging code, views can be added or removed from any perspective to aid productivity by keeping the most commonly used views in one perspective. For instance, as most development takes place in the C/C++ perspective the common views needed for running the application (e.g. resource manager and output console) can be added to this development perspective to avoid continually switching perspectives (see Figure 15).

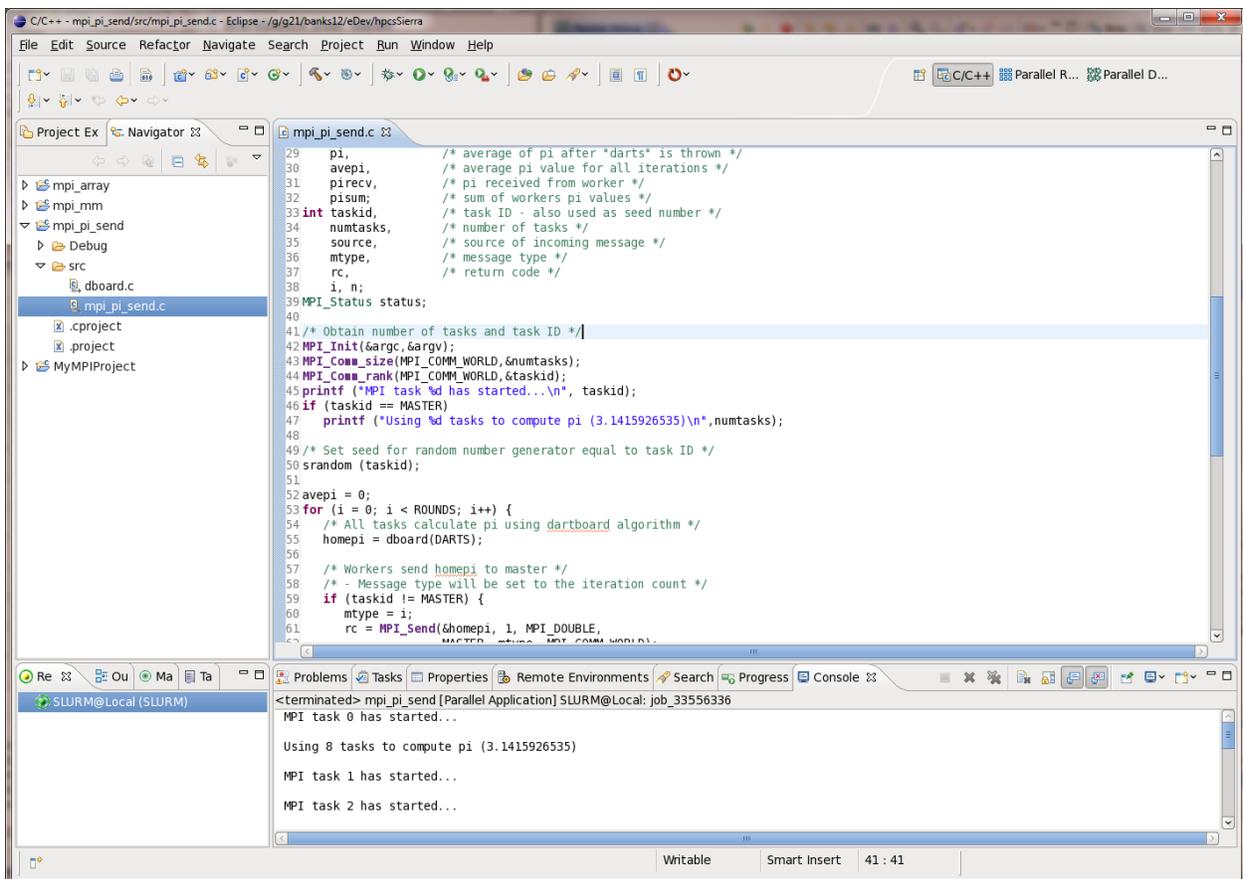


Figure 15: Tailored perspective with additional views.

## 6 Debugging

### 6.1 Description

Debugging is a key phase of the development process. Methods of debugging have evolved from inserting print statements at key locations in code to examining the contents of variables at given points

in the execution to using a full blown debugger that allows interactive examination of all variables at each execution step. Typically a debugger is applied to a single process, but on a parallel cluster the debugger must be able to monitor and control the processes and threads for each node on which the parallel job is running. The PTP bundle includes the Scalable Debug Manager (SDM) which consists of a front-end client plugin to Eclipse, the GNU gdb backend debug engine, and a communication manager which marshals debug messages within a tree-based network connecting each running process.

## 6.2 User Experience

To start a debug session for a project, a debug launch configuration must be created. From the main menu bar, select Run|Debug Configurations. This action brings up a window nearly identical to the Run Configuration with the inclusion of a Debugger tab (see Figure 16). The current Run Configurations are under Parallel Applications. Simply create a copy of a run configuration, give this configuration a new name; a good convention is to simply append '-debug' to the run name, and modify this configuration to invoke a debug session.

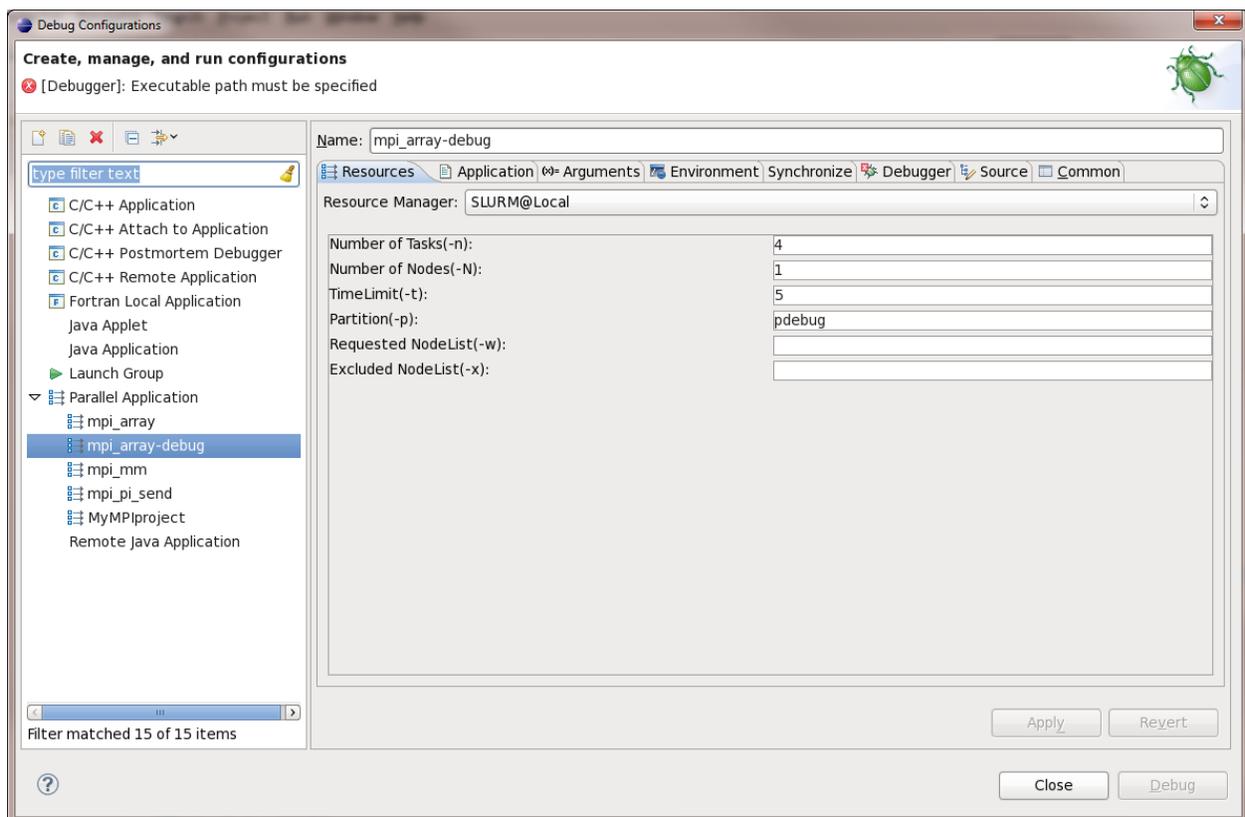


Figure 16: Debug Configuration

It is common for a cluster to have a set of nodes dedicated for debugging use. This debug partition should be specified as the partition to use in this configuration. Note that the icon on the Debugger tab indicates an error in the configuration and the message at the top of the panel states that the path to the debug executable must be specified. Clicking on the Debugger tab allows adding the path to the

## PTP v5.0.3 Evaluation

SDM executable which can be found in the Eclipse installation plugin directory:  
plugins/org.eclipse.ptp.debug.sdm\_5.0.3.201110141146/sdm.

Clicking the Debug button will launch the project and bring up the *Parallel Debug perspective* (see Figure 17).

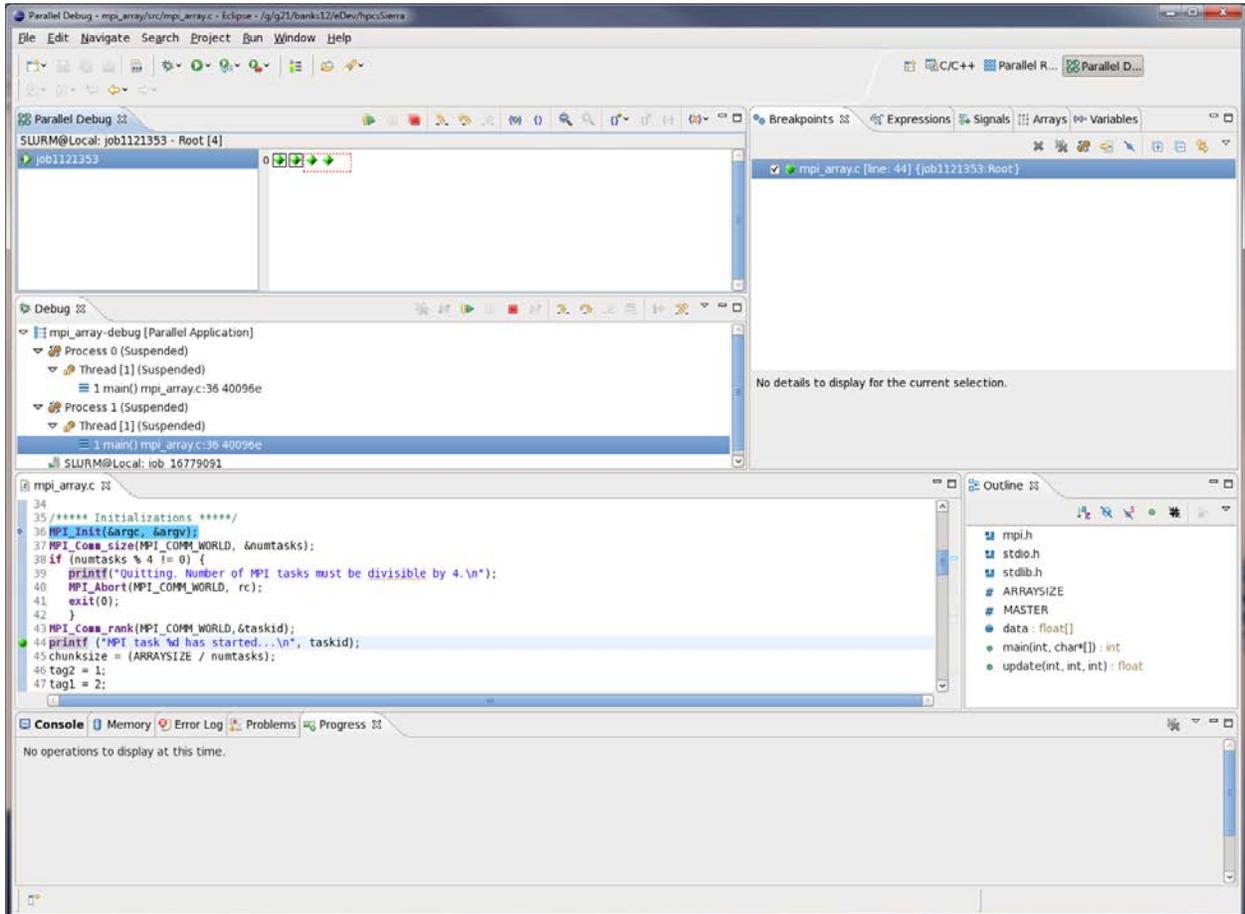


Figure 17: Parallel Debug Perspective.

As is typical with a debug interface, this perspective is highly interactive and packed with information concerning the running processes. The Parallel Debug view shows the job and the associated processes (small diamonds icons). In this view, processes can be grouped into arbitrary sets allowing logical process groupings to be examined independently. When initiated, all processes are contained in a *root* set. The creation of sets is accomplished by *mouse-dragging* a rectangle over the processes (see red rectangle in Parallel Debug view in Figure 17), and/or conventional *ctrl-clicking* to select individual processes. Any number of sets can be created and uniquely named. Processes can be added or removed from a set during the debug session. Only one set at a time will be displayed and operated on by the debugger.

Individual processes, whether in a set or not, can be *registered* with the debugger to enable the examination of its stack frames and threads. A registered process is indicated by a square around the individual process diamond. The Debug view shows the registered processes and their associated threads. Clicking on a process will change the context of the other views to that process. For instance, the code view is associated with the selected process as well as the Breakpoints, Expressions, Signals, Arrays, and Variables.

In the source code view, two types of breakpoints can be created at specific points in the code. A *global breakpoint* will apply to all processes in the job. With no other breaks specified, each process will continue execution to this type of break point. A global breakpoint is typically used to bring all processes to a common break at which point the developer can begin to drill down into specific logic in specific processes. This drill-down debugging is done by setting a *set breakpoint* which will only be applied to an individual process set.

### 6.3 Evaluation

Stepping through parallel processes in a SDM debug session takes a bit of getting used to, but once familiar, the tool is quite powerful in showing the individual process flow as well as process interaction. Using global breakpoints is straightforward in that all processes will execute until the breakpoint is reached. Sets and set breakpoints may be created only after the job is submitted and are not retained from one run to the next. For this reason the ‘stop in main’ option should always be selected in the debug configuration to ensure the suspension of execution after processes have been initialized.

Setting and keeping track of breakpoints for individual sets and processes rely heavily on the visual information provided in each of the views (refer to Figure 18). First it’s important to ensure that each process to be examined is registered – as indicated by the *selected* process diamonds in the Parallel Debug view. The Debug view shows the current stack trace of each thread in each registered process. The state each thread is in is shown (e.g. suspended, running, etc.) and if suspended, the source code line number is given. Selecting a specific process thread in this view allows stepping through the execution of the selected thread while all other processes remain suspended. If the thread reaches a ‘running’ state and trace step icons become disabled (i.e. greyed-out), then it’s likely that the process has reached a dependency point with another suspended process.

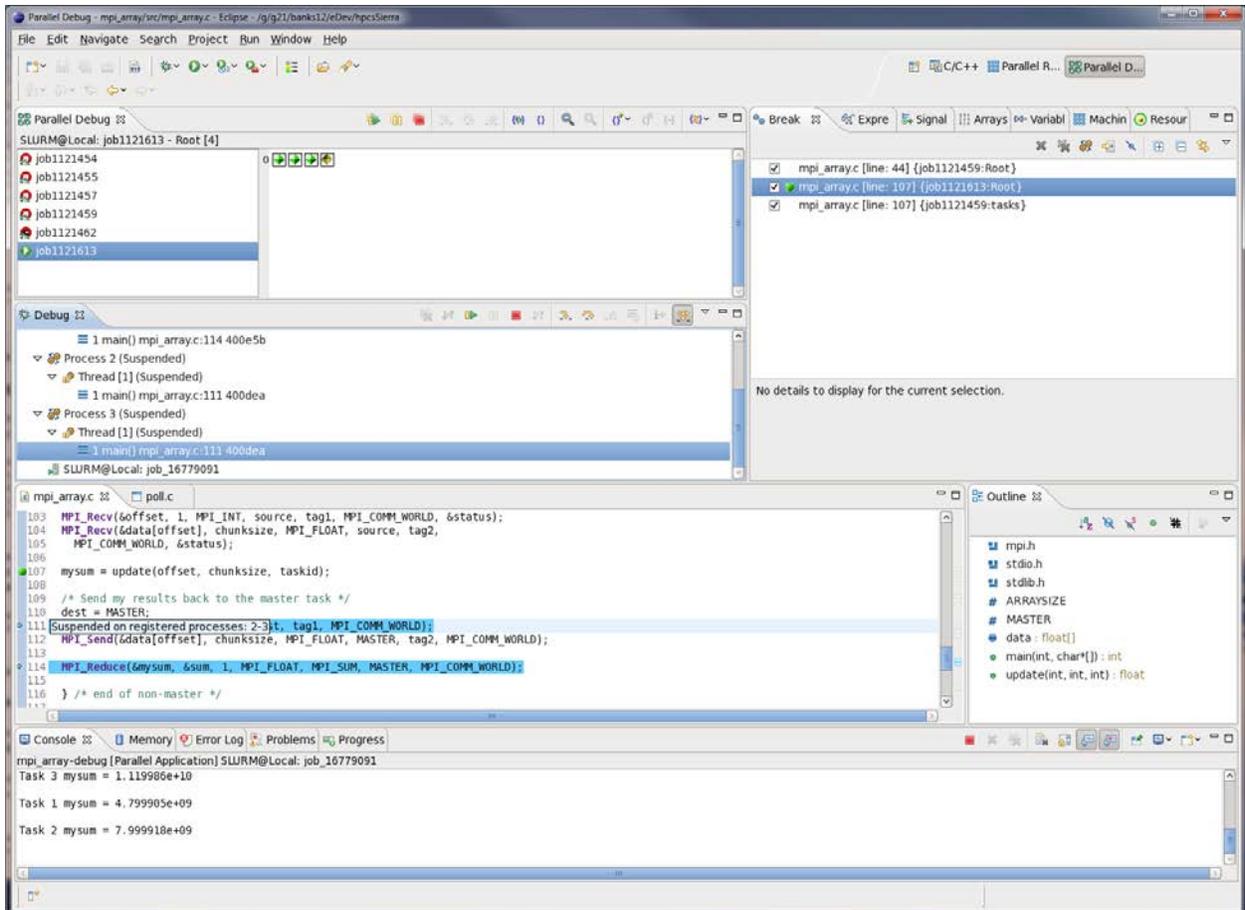


Figure 18: Debug session in progress.

The source code view highlights the source statements for the execution trace of all registered processes in the selected set. Hovering over the break marker, to left of the line number, shows which processes are currently suspended at that statement.

Adding other views to this perspective can be quite beneficial. For instance, bringing in the Machine view allows drill down on the job execution nodes (see Figure 19). This view in turn shows the jobs and processes running on that node. Drilling into the properties of a specific process shows the process details and output generated by just that process.

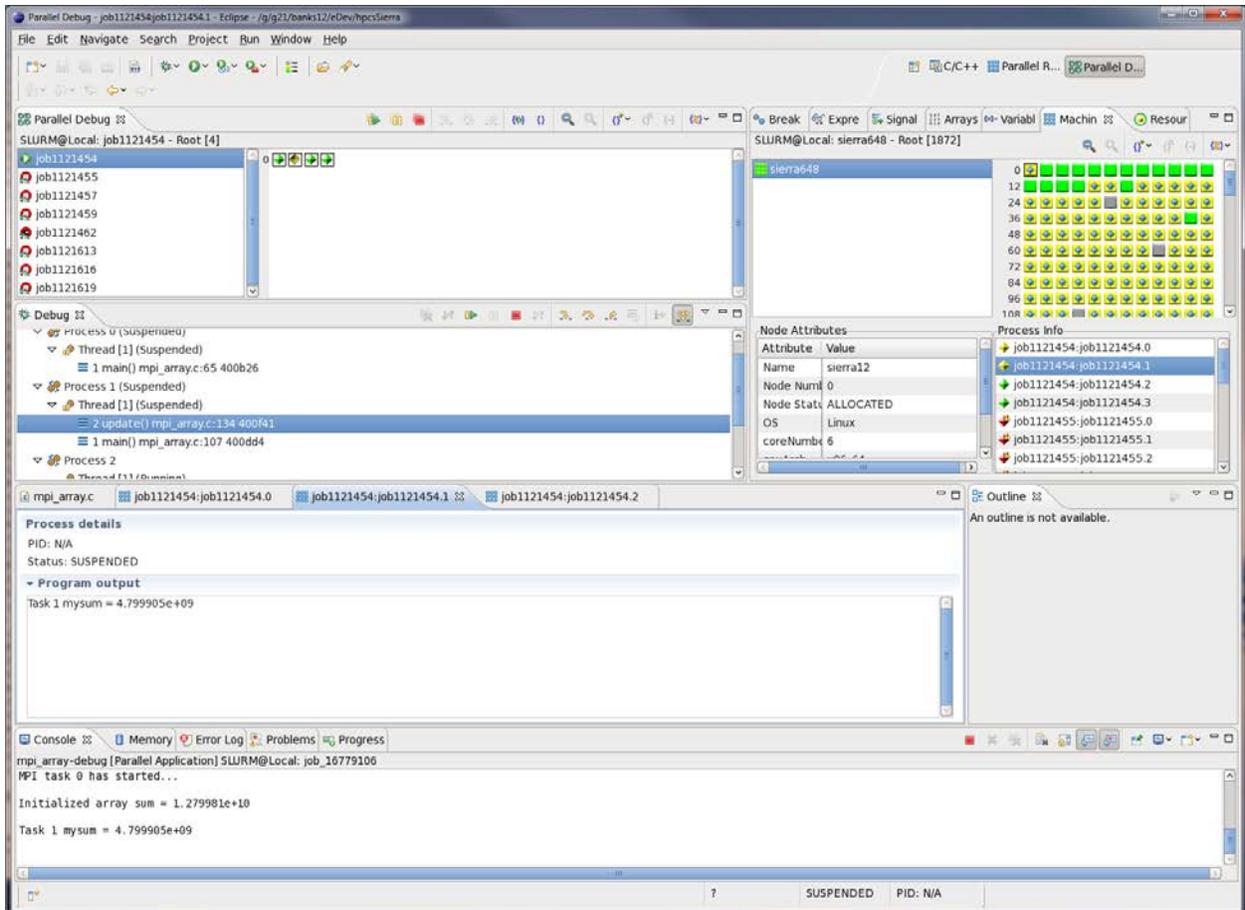


Figure 19: Individual process views for debugging.

An alternative debugger available here at LLNL is the TotalView debugger available from Rogue Wave Software [6]. Like the SDM client, the TotalView interface consists of multiple windows, or views, each focused on specific functionality for a debug session. Besides the basic functionality found in PTP's SDM, TotalView provides several advanced features including memory tracking, and support of C++ template libraries.

As the SDM debugger continues to mature it may begin to include some of these more advanced features. However, the basic functionality currently integrated into the PTP plugin is well presented in the interface and provides a quick and easy means for an iterative code/debug cycle.

## 7 Evaluation Summary

The key advantage of any IDE is the Integration. The editing of source code is integrated with the compiler allowing real-time feedback on code syntax. Integration with the linker ensures adherence to the protocol of included or created library functions. Developing source code for a project is, and should be, a very iterative process. All software developers will create a subset of desired basic functionality, execute the code to test/debug, add more functionality – test/debug, etc. This cycle is repeated multiple

times until all functional requirements are met, and the code executes flawlessly. This cyclic process is where the Eclipse IDE shines in increasing the productivity of the developer.

The Parallel Tools Platform (PTP) is a plug-in module for Eclipse that aids in the development of parallel applications. PTP incorporates the popular CDT (C/C++ Development Tooling) module to provide a source code editor with color-coded syntax highlighting, context sensitive code completion, and compile-time error checking. PTP includes the SDM debugging tool for parallel applications, and utility tools to submit and monitor jobs on a parallel system.

### **Installation**

By necessity the installation is more involved than simply pointing Eclipse to an update site and downloading a plugin. Several utilities must be built from source on the parallel host system in order to ensure proper executing on each specific cluster. The included documentation was well done, presenting step-by-step instructions for installing each utility. As a side note, at the beginning of this evaluation an earlier version of PTP was installed and subsequently upgraded to v5.0.3. The upgrade documentation did an excellent job of walking through the steps necessary to rebuild these utilities and re-pointing the IDE preferences to use the newly built version.

### **Environment Configuration**

Configuring the ToolChain for the desired compiler, linker, and libraries is a bit frustrating. More standard configurations could be provided in the default set. More importantly the ability to add a new configuration as opposed to overriding an existing default should be available.

### **Code Development**

The process of creating projects, editing/importing code and running an executable exceeded expectations. The use of a proxy to communicate with the actual resource manager proved to make running on a parallel cluster nearly seamless as compared to running on a standalone node. While there was the extra complexity of having to build, configure, and start the proxy, the benefit was well worth the effort. The proxy allows the integration of the machine configuration, parallel node status and monitoring of each running process to simply be included in another view of the IDE.

### **Debugging**

The SDM debugging tool has a learning curve to understand how to step through specific processes, with basic debugging functionality present. The ability to drill into a process starting at the node view is a nice feature. While more complicated issues with highly parallel code may be better debugged with the additional features provided in a debugger such as TotalView, the SDM debugger has the necessary capabilities to step through individual threads examining the variables and behavior along the way. For this integrated functionality, it's hard to beat the free price.

## 8 References

1. **Gregory R. Watson, Craig E. Rasmussen.** *A Strategy for Addressing the Needs of Advanced Scientific Computing Using Eclipse as a Parallel Tools Platform.* Los Alamos : Los Alamos National Laboratory, 2005. LA-UR-05-9114.
2. **anonymous.** PTP - Parallel Tools Platform. *Eclipse.org.* [Online] Eclipse Foundation. [Cited: November 7, 2011.] <http://eclipse.org/ptp/>.
3. **Livermore Computing Services.** Open Computing Facility - OCF. *High Performance Computing @ Lawrence Livermore National Laboratory.* [Online] Livermore Computing Services, 12 07, 2011. [Cited: 12 10, 2011.] [https://computing.llnl.gov/?set=resources&page=OCF\\_resources#sierra](https://computing.llnl.gov/?set=resources&page=OCF_resources#sierra).
4. Parallel Tools Platform (PTP) User Guide. *Eclipse Online Help.* Ottawa : Eclipse Foundation, Inc., 2011.
5. CDT Project. *Eclipse.org.* [Online] Eclipse Foundation. [Cited: January 3, 2012.] <http://www.eclipse.org/cdt/>.
6. Parallel Tools Platform (PTP) Wiki page. *PTP - Eclipsepedia.* [Online] Eclipse Foundation. [Cited: 11 7, 2011.] <http://wiki.eclipse.org/PTP>.
7. **Rogue Wave Software.** TotalView. *roguewave.com.* [Online] Rogue Wave Software. [Cited: December 10, 2011.] <http://www.roguewave.com/products/totalview.aspx>.