The Eclipse Parallel Tools Platform and Scientific Application Development

Beth Tibbitts, IBM  tibbitts@us.ibm.com
Greg Watson, IBM  g.watson@computer.org
Craig Rasmussen, LANL  crasmussen@lanl.gov

The most recent version of these tutorial slides will be available at http://eclipse.org/ptp

“This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under its Agreement No. HR0011-07-9-0002”

11/09/07
## Tutorial Outline (morning)

<table>
<thead>
<tr>
<th>Time</th>
<th>Module</th>
<th>Outcomes</th>
<th>Presenter</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30 – 8:45</td>
<td>Tutorial Introduction</td>
<td>✦ Overview of the tutorial process and setup</td>
<td>Greg Watson</td>
</tr>
<tr>
<td>8:45 - 9:15</td>
<td>1. Overview of Eclipse and PTP</td>
<td>✦ An understanding of the overall Eclipse and PTP architecture</td>
<td>Greg Watson</td>
</tr>
<tr>
<td>9:15 - 10:00</td>
<td>2. Installing Eclipse</td>
<td>✦ Eclipse installed on your laptop</td>
<td>Beth Tibbitts</td>
</tr>
<tr>
<td>10:00 - 10:30</td>
<td>Break</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:30 - 11:30</td>
<td>3. Introduction to the Eclipse IDE</td>
<td>✦ Knowledge of the basic features of the Eclipse IDE</td>
<td>Craig Rasmussen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✦ Building, running and debugging a sample application</td>
<td></td>
</tr>
<tr>
<td>11:30 - 12:00</td>
<td>4. Advanced Development</td>
<td>✦ Knowledge of some of the advanced features of the Eclipse IDE</td>
<td>Craig Rasmussen</td>
</tr>
<tr>
<td>12:00 - 1:30</td>
<td>Lunch Break</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Tutorial Outline (afternoon)

<table>
<thead>
<tr>
<th>Time</th>
<th>Module</th>
<th>Outcomes</th>
<th>Presenter</th>
</tr>
</thead>
</table>
| 1:30 - 3:00   | 5. PTP and Parallel Language Development Tools | ✦ Introduction to PTP  
✦ Creating and launching a parallel application  
✦ Experience using PLDT tools on a real application | Beth Tibbitts     |
| 3:00 - 3:30   | Break                                       |                                                                          |                   |
| 3:30 - 4:30   | 6. Parallel Debugging                       | ✦ Introduction to the Eclipse parallel debugger, locating and correcting bugs in parallel code | Greg Watson       |
| 4:30 - 4:50   | 7. Where To Go Next                         | ✦ Further information about Eclipse, PTP and related tools               | Greg Watson       |
| 4:50 - 5:00   | Tutorial Wrap Up                            | ✦ Completed feedback forms                                               | Greg Watson       |
A word on versions...

- The current release of PTP (1.1.1) requires CDT version 3.1.x and Eclipse 3.2.x. (Callisto)
- The next release of PTP (2.0) will require CDT 4.0 and Eclipse 3.3 (Europa) which were released in June 2007
  - PTP 2.0 will not be released until the end of 2007
- The slides in this tutorial describe an early access version of PTP 2.0, which requires Eclipse 3.3.1.1 and CDT 4.0.2, so that you can see the latest features that will be available
  - There may be some minor differences between PTP that you see here and the final release version
Module 1: Overview of Eclipse and PTP

- **Objective**
  - To introduce participants to the Eclipse platform and PTP

- **Contents**
  - History
  - What is Eclipse?
  - Who is using Eclipse?
  - What is PTP?
History

- Originally developed by Object Technology International (OTI) and purchased by IBM for use by internal developers
- Released to open-source community in 2001, managed by consortium
  - Eclipse Public License (EPL)
  - Based on IBM Common Public License (CPL)
- Consortium reorganized into independent not-for-profit corporation, the Eclipse Foundation, in early 2004
  - Participants from over 100 companies
Eclipse Foundation

- Board of Directors drawn from four classes of membership:
  - Strategic Developers, Strategic Consumer, Add-in Providers, and Open Source project leaders
- Full-time Eclipse management organization
- Councils guide the development done by Eclipse Open Source projects
  - Requirements
  - Architecture
  - Planning
- Currently 9 projects and over 50 subprojects
Members of Eclipse
June 2007

- 162 members in June ’07 (130 in March 2006)
- 21 strategic members (16 in June 2006)
- 794 committers, representing 48 organizations
What is Eclipse?

- A vendor-neutral open source development platform
- A universal platform for tool integration
- Plug-in based framework to create, integrate and utilize software tools
Equinox

- OSGi framework implementation model
  - Formerly known as the Open Services Gateway initiative
  - Standard for application lifecycle management
- Provides the most fundamental Eclipse infrastructure
  - Plug-ins (known as a bundle)
  - Bundle install, update and uninstall
  - Bootstrap and launching
  - Extension registry
- Introduced in Eclipse 3.0
Platform

- Core frameworks and services with which all plug-in extensions are created
- Represents the common facilities required by most tool builders:
  - Workbench user interface
  - Project model for resource management
  - Portable user interface libraries (SWT and JFace)
  - Automatic resource delta management for incremental compilers and builders
  - Language-independent debug infrastructure
  - Distributed multi-user versioned resource management (CVS supported in base install)
  - Dynamic update/install service
Plug-ins

- Java Development Tools (JDT)
- Plug-in Development Environment (PDE)
- C/C++ Development Tools (CDT)
- Parallel Tools Platform (PTP)
- Fortran Development Tools (Photran)
- Test and Performance Tools Platform (TPTP)
- Business Intelligence and Reporting Tools (BIRT)
- Web Tools Platform (WTP)
- Data Tools Platform (DTP)
- Device Software Development Platform (DSDP)
- Many more...
Module 2: Installing Eclipse

✦ Objective
  ✦ To learn how to install Eclipse
  ✦ To install Eclipse on your laptop

✦ Contents
  ✦ Software prerequisites
  ✦ Installing Eclipse
  ✦ Installing CDT, RSE and PTP
Software Prerequisites

- Java (1.5 or later)
- Cygwin (for Windows)
- make, gcc, and gdb (or other vendor compilers)
- gfortran (only required for Fortran support)
- OpenMPI or MPICH2 (only required for PTP Runtime)
## Pre-installation Overview

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>Cygwin</th>
<th>make/gcc/gdb</th>
<th>gfortran</th>
<th>OpenMPI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Windows</strong></td>
<td>install</td>
<td>install</td>
<td>installed by cygwin</td>
<td>install</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Linux</strong></td>
<td>install</td>
<td>-</td>
<td>check installed</td>
<td>install</td>
<td>install</td>
</tr>
<tr>
<td><strong>MacOS X</strong></td>
<td>-</td>
<td>-</td>
<td>requires Xcode</td>
<td>install</td>
<td>install</td>
</tr>
</tbody>
</table>
Java Installation

- Eclipse requires Sun or IBM versions of Java
  - Only need Java runtime environment (JRE)
  - Java 1.5 is the same as JRE 5.0
  - The GNU Java Compiler (GCJ) will not work!
- Latest Sun JRE is in the java folder on tutorial CD:
  - jre-1_5_0_12-windows-i586-p.exe
  - jre-1_5_0_12-windows-amd64.exe
  - jre-1_5_0_12-linux-i586.bin
  - jre-1_5_0_12-linux-amd64.bin
Java Installation (Linux)

- Open a terminal window
- Mount your CDROM if necessary

```
mount /media/cdrom
```

- Enter the commands below:
  - Replace `cdrom` with the location of your CDROM (usually `/media/cdrom`) and `arch` with your computer architecture (usually `i586`)
  ```
cd
  cdrom/java/jre-1_5_0_12-linux-
arch.bin
  ```
  - Hit space until you are asked to agree to license, then enter ‘yes’)
  ```
  PATH=~/jre1.5.0_12/bin:$PATH
  ```
- Add to your PATH in your login file if required
Java Installation (Windows)

- Open the **TutorialCD** in **My Computer**
- Open the **java** folder
- Double-click on **jre-1_5_0_12-windows-arch**
  - Replace **arch** with your computer architecture (most likely **i586-p**)
- Follow installer wizard prompts
  - Accept default options
Eclipse and PTP Installation

- Eclipse is installed in two steps
  - First, the ‘base’ Eclipse is downloaded and installed
    - This provides a number of pre-configured ‘features’
  - Additional functionality is obtained by adding more ‘features’
    - This can be done via an `update site’ that automatically downloads and installs the features
    - Features can also be downloaded and manually installed
- PTP requires the following features
  - C/C++ Development Tools (CDT)
  - Remote System Explorer (RSE)
  - Parallel Tools Platform (PTP)
### Eclipse and PTP Installation Overview

<table>
<thead>
<tr>
<th></th>
<th>Eclipse SDK</th>
<th>CDT Feature</th>
<th>RSE Feature</th>
<th>PTP Feature</th>
<th>PTP Proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Windows</strong></td>
<td>install</td>
<td>update</td>
<td>update</td>
<td>update</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Linux</strong></td>
<td>install</td>
<td>update</td>
<td>update</td>
<td>update</td>
<td>install</td>
</tr>
<tr>
<td><strong>MacOS X</strong></td>
<td>install</td>
<td>update</td>
<td>update</td>
<td>update</td>
<td>install</td>
</tr>
</tbody>
</table>
Eclipse SDK Installation

- The base component of Eclipse is known as the **Eclipse SDK**
- The Eclipse SDK is downloaded as a single zip or gzipped tar file
- You must have the correct file for your operating system and windowing system
- Unzipping or untaring this file creates a directory containing the main executable
- Copies of the Eclipse SDK for each operating system type are located in the **eclipse** folder on the tutorial CD
Eclipse SDK Installation (Linux)

- Open a terminal window
- Mount CDROM if not already
- Enter the commands below:
  - Replace `cdrom` with the location of your CDROM (usually `/media/cdrom`)
  - If your machine is not x86 based, use either the `-ppc` or `-x86_64` versions (not on CDROM)

```
cd
tar -zxvf cdrom/eclipse/eclipse-SDK-3.3.1.1-linux-gtk.tar.gz
```
Eclipse SDK Installation (MacOS X)

- From the Finder, open **TutorialCD**
- Open the **eclipse** folder
- Double-click on `eclipse-SDK-3.3.1.1-macosx-carbon.tar.gz`
- Will create new eclipse folder in your **downloads** location
  - Specified in Safari
- Drag new **eclipse** folder to **Applications** (or wherever you want to install it)
Eclipse SDK Installation (Windows)

- Open the TutorialCD in My Computer
- Open the eclipse folder
- Unzip the following file:
  `eclipse-SDK-3.3.1.1-win32.zip`
- Choose a location on your hard drive where you want to install Eclipse (e.g. C: \)
  - An eclipse folder will be created at this location
Starting Eclipse

- **Linux**
  - From a terminal window, enter
  ```
  cd
eclipse/eclipse &
  ```

- **MacOS X**
  - From finder, open the **Applications** ➤ **eclipse** folder
  - Double-click on the **Eclipse** application

- **Windows**
  - Open the **eclipse** folder
  - Double-click on the **eclipse** executable

- Accept default workspace when asked
- Select workbench icon from welcome page
Adding Features

✧ New functionality is added to Eclipse using features
✧ Features are obtained and installed from an update site (like a web site)
✧ Features can also be installed manually by copying files to the features and plugins directories in the main eclipse directory
Installing Eclipse Features from an Update Site

- Three types of update sites
  - Remote - download and install from remote server
  - Local - install from local directory
  - Archived - a local site packaged as a zip or jar file

- Eclipse 3.3.1 comes preconfigured with a link to the Europa Discovery Site
  - This is a remote site that contains a large number of official features
  - Europa projects are guaranteed to work with Eclipse 3.3.1

- Many other sites offer Eclipse features
  - Use at own risk
Creating a Local Update Site

- We have combined everything needed for the tutorial onto a local update site on the CDROM.
- From the Help menu, choose Software Updates → Find and Install...
- Select Search for new features to install.
- Click Next >
- Click New Local Site...
- Navigate to your CDROM, select the updatesite folder and click Choose (OK on Linux).
- Enter Tutorial for the Name.
- Click OK.
Installing Tutorial Features

- Make sure only **Tutorial** is selected, other options as defaults
- Click **Finish**
- From **Search Results**, select **Tutorial** (open the twisty to see the contents)
- Click **Next >**
- Accept the license terms
- Click **Next >**
- Click **Finish**
- For **Feature Verification**, click **Install All**
- Restart workbench when asked
Installing the PTP Proxy  
(for information only)

- Normally installed on a parallel machine
  - e.g. a cluster
  - Can install on a non-parallel system
- Not available for Windows
- Requires OpenMPI to be built and installed
  - This process depends on the type of machine
  - Beyond the scope of this tutorial
- To install the proxy, do the following steps from a terminal
  - Change to your Eclipse installation directory
  - Change to `plugins/org.eclipse.ptp.os.arch_2.0`, where
    `os` is your operating system (macosx or linux), `arch` is your architecture (ppc, x86, or x86_64)
  - Run the command: `sh BUILD`
Module 3: Introduction to the Eclipse IDE

- **Objective**
  - Gain an understanding of how to use Eclipse to develop applications

- **Contents**
  - Brief introduction to the Eclipse IDE
  - Create a simple application
  - Run and debug simple application
Platform Differences

- Single button mouse (e.g. MacBook)
  - Use Control-click for right mouse / context menu
- Context-sensitive help key differences
  - Windows: use F1 key
  - Linux: use Shift-F1 keys
  - MacOS X
    - Full keyboard, use Help key
    - MacBooks or aluminum keyboard, create a key binding for Dynamic Help to any key you want
- Accessing preferences
  - Windows & Linux: Window ➤ Preferences...
  - MacOS X: Eclipse ➤ Preferences...
Specifying A Workspace

- Eclipse prompts for a workspace location at startup time
- The workspace contains all user-defined data
  - Projects and resources such as folders and files

The prompt can be turned off
Eclipse Welcome Page

- Displayed when Eclipse is run for the first time

Select “Go to the workbench”
The Workbench represents the desktop development environment:

- It contains a set of tools for resource management.
- It provides a common way of navigating through the resources.
- Multiple workbenches can be opened at the same time.
A Workbench contains perspectives
A Perspective contains views and editors
Perspectives

- Perspectives define the layout of views in the Workbench.
- They are task oriented, i.e. they contain specific views for doing certain tasks:
  - There is a Resource Perspective for manipulating resources.
  - C/C++ Perspective for manipulating compiled code.
  - Debug Perspective for debugging applications.
- You can easily switch between perspectives.
Switching Perspectives

- You can switch Perspectives by:
  - Choosing the **Window ➤ Open Perspective** menu option
  - Clicking on the **Open Perspective** button
  - Clicking on a perspective shortcut button
Available Perspectives

- By default, certain perspectives are available in the Workbench
- We’ve also installed C/C++ perspective
Customizing Perspectives

- Items such as shortcuts, menu items and views may be customized
  - **Window ▶ Customize Perspective…**
- Rearrange views by dragging
  - Try moving the outline view
- Save changes
  - **Window ▶ Save Perspective As…**
- Close Perspective
  - Right-click on perspective title and select **Close**
- Reset Perspective
  - **Window ▶ Reset Perspective** resets the current perspective to its default layout
Views

- The main purpose of a view is:
  - To provide alternative ways of presenting information
  - For navigation
  - For editing and modifying information

- Views can have their own menus and toolbars
  - Items available in menus and toolbars are available only in that view
  - Menu actions only apply to the view
Stacked Views

- Stacked views appear as tabs
- Selecting a tab brings that view to the foreground
Project Explorer View

- Represents user’s data
- It is a set of user defined resources
  - Files
  - Folders
  - Projects
    - Collections of files and folders
    - Plus meta-data
- Resources are visible in the Project Explorer View
Opening a New View

- To open a view:
  - Choose **Window ➤ Show View ➤ Other…**
  - The **Show View** dialog comes up
  - Select the view to be shown
  - Select **OK**
Fast Views (1)

- Hidden views that can be quickly opened and closed
  - They take up space in the Workbench
- Fast views can be created by:
  - Dragging an open view to the shortcut bar
  - Selecting **Fast View** from the view’s menu
- A Fast View is activated by clicking on its **Fast View** button

Outline view has been hidden in the shortcut bar
Fast Views (2)

- Clicking on the Fast View opens the view in the current perspective
- Clicking outside of the view makes it hidden again
- Turn off the Fast View by selecting Fast View from the view’s menu again
Editors

- An editor for a resource opens when you double-click on a resource
- The type of editor depends on the type of the resource
  - .c files are opened with the C/C++ editor
  - Some editors do not just edit text
- When an editor opens on a resource, it stays open across different perspectives
- An active editor contains menus and toolbars specific to that editor
- When you change a resource, an asterisk on the editor’s title bar indicates unsaved changes
Source Code Editors

- A source code editor is a special type of editor for manipulating source code
- Language features are highlighted
- Marker bars for showing
  - Breakpoints
  - Errors/warnings
  - Tasks
- Location bar for navigating to interesting features
Preferences

- Preferences provide a way for you to customize your Workbench
  - By selecting **Window ➤ Preferences...** or **Eclipse ➤ Preferences...**

- Examples of preference settings
  - Use Emacs bindings for editor **keys**
  - Modify editor folding defaults
    - E.g., fold all macro definitions
  - Associate file types with file extensions
    - E.g., *.f03 with the Fortran editor
  - Toggle automatic builds
  - Change key sequence shortcuts
    - E.g., Ctrl+/

Help

- Access help
  - Help ▶ Help Contents
  - Help ▶ Search
  - Help ▶ Dynamic Help
- Help Contents provides detailed help on different Eclipse features
- Search allows you to search for help locally, or using Google or the Eclipse web site
- Dynamic Help shows help related to the current context (perspective, view, etc.)
Creating A Simple Application

Outline:
- Create a C Project
- Add files
  - Source files (ending in .c)
  - A makefile is automatically created
- Build application
  - Done automatically
- Debug application
  - Create a Debug Configuration
CDT Projects

- A **Project** contains the resources of an application
- Projects and their resources are visible in **Project Explorer view**
- Some resources are “smart”
  - **Binaries** - collects all project executables together
  - **Includes** - shows all included files, including system files
  - **Archives** - collects all project libraries together
Project Types

- **Project Type is very important**
  - Determines how the project will be built
  - Selects the toolchain used to build the project
  - Defines the resulting object

- **There are two main types of projects**
  - Automatic makefile generation
    - **Executable** - an ordinary executable binary
    - **Shared Library** - a shared library that can be dynamically loaded
    - **Static Library** - a static library that can be linked into an application executable
  - Externally supplied makefiles
    - **Makefile project** - builds anything the makefile specifies
Creating a C Project

- Make sure the C/C++ Perspective is selected
- Choose **File > New > C Project** or select drop down next to **New Project** button then **C Project**
- Give it a name: e.g. Zproject
- Select a project type from the list of **Project types** (default is OK)
- Click **Next >**
Selecting Configurations

- A **Configuration** allows you to customize the project for deployment on a particular platform.
- By default, CDT will create **Debug** and **Release** configurations.
- You can choose which configuration to use when launching the application.
- You also have the chance to manually set project properties by clicking on **Advanced Settings**.
- Select **Finish** to complete project creation.
Adding Resources

Resources can be added to a project by:
- Creating new resources
- Importing existing resources from another location

We will import existing files from file system
- Right-click on project, select Import...
- Open the General folder and select File System
- Click Next >
Importing Resources

- Click **Browse**...
- Navigate to and select the **samples** folder on the tutorial CD
- Click on the **samples** folder
- Select check box next to **linear_function.c** and **testz.c**
- Click **Finish**
Outline View

- Workbench now shows project files
- Double-click on testz.c source file in the **Project Explorer** to open C editor
- Outline view is shown for file in editor
Fix Error in File

- Build project: select build icon on toolbar:
  - Project fails to build
  - Note red icon on filename
- Click on **Problems** View tab
- Fix error in **linear_function.c**
  - Double-click on the file in the **C/C++ Projects** view to open an editor
- Save file and rebuild project
  - **File** » **Save** (or Ctrl-S)
  - Select the build icon on the toolbar.
- Look at console view to see build progress
  - There is still another error
To fix the next error, the required library must be added to the build process.

This is done by editing the **Linker** tool settings in the **C/C++ Build** properties for the project.

Right-click on the project and select **Properties** menu item.

Select the **C/C++ Build** item. Under that, select the **Settings** item.
Adding A Library

- From the **Tool Settings** tab:
  - Windows Cygwin select **Cygwin C Linker** → **Libraries**
  - Linux/Mac select **GCC C Linker** → **Libraries**
- Click on ‘+’ icon next to **Libraries (-l)** to add library
- Enter ‘z’ in the dialog box and select **OK**
- Select **OK** to close the **Project Properties**
- Rebuild project; there should be no errors
Launch Configuration

- A Launch Configuration is needed to run or debug an application
- This contains all the information associated with the execution
  - Command-line arguments
  - Environment variables
  - Debugger options
  - Configuration to launch
- All this information is remembered to make re-launching the application simple
A Debug Launch Configuration

- Select Run ➔ Open Debug Dialog…
- Select C/C++ Local Application
- Click the New Launch Configuration button
- Everything should be already configured for the launch
- Click the Debug button to launch
- Select Yes to confirm switching to the Debug Perspective after launching

*If cygwin (Windows) gives source path lookup error, see next slide*
Windows Cygwin debugging with CDT

From http://wiki.eclipse.org/CDT/User/FAQ
- look under Debugging:
I'm using cygwin and when launching the debugger, it complains that it can't find the source file

- You must provide a mapping from /cygdrive/c to c:\ (or whatever your drive letter is).
- To do this,
  - From the editor error page, select the "Edit Source Lookup Path..." button and select the "Add..." button
    - Or, in the eclipse IDE, go to menu Window -> Preferences -> C/C++ -> Debug-> Common Source Lookup Path -> Add.
  - From the list of lookup containers, choose Path Mapping and OK. You get a New Mapping in the list.
  - Select the mapping and then Edit. In the Modify the path mappings dialog, select Add, and then enter:
    - /cygdrive/c as the compilation path and
    - c:\ as the local file system path.
  - Select OK, OK, OK to finish the dialogs.
- Terminate the debug session and restart; it should find your source files now.
- This setting will apply to any debug sessions launched from this workspace.
- You can also modify the settings in each individual launch configuration.
Debug Perspective

- Controls for resuming, stepping, terminating, etc.
- **Debug view** shows stack frames and threads
- **Source view** shows current line marker and breakpoints
- **Variables view** shows values of local variables
- **Console** shows program output
Debugging (1)

- Set a breakpoint by double-clicking on the left vertical bar in the editor at $x = x + \text{myFn}(x)$ line
- To continue running, click on Resume button
- Click on Step Into button to enter myFn()
Debugging (2)

- Examine variables in **Variables view**
  - Clicking on a variable will display its value
- Select a different stack frame in the **Debug view** if desired
- Back in the top stack frame, click on the **Step Return** button
- Finish by clicking on the **Resume** or **Terminate** button
Module 4: Advanced Development

- **Objective**
  - Create and build a Standard Make Project from source files in CVS

- **Contents**
  - Version control
  - Standard Make Projects
  - C/C++ Projects
  - Task Tags, Bookmarks
  - Refactoring
  - Searching
Version Control (CVS)

- Version control provided through the **Project Explorer View**, in the **Team** context menu
- Provides familiar actions:
  - Commit...
  - Update...
- Also less used tasks:
  - Create/Apply Patch...
  - Tag as Version
  - Branch...
  - Merge...
  - Add to .cvsignore...
Add Repository Location

- Select **Window ➤ Open Perspective ➤ Other...**
- Select **CVS Repository Exploring** then **OK**
- Right-click in **CVS Repositories View**, then select **New ➤ Repository Location...**
- Set **Host** to the IP address of remote machine
- Set **Repository path** to `/home/YOUR_USERNAME`
- Fill in **Username** and **Password**
- Set **Connection type** to **extssh**
- Check **Save password**
- Select **Finish**
Checkout Project Code

- Open the repository, then open HEAD
  - Right-click on **MyCVSProject** ➤ **Check out As…**
    - Make sure “Check out as a project configured using the New Project Wizard” is selected
  - Select **Finish**
  - Select **C/C project**
  - Select **Next>**
- Enter **Project name** (MyCVSProject) and **location**
  - Can put project in location other than workspace
- Under **Project Types**, select **Makefile project**
  - Ensures that CDT will use existing makefiles
- Select **Finish**
- Switch to the **C/C++ Perspective**
About Makefiles and autoconf

- Can create project Makefiles with the Makefile Editor
  - Syntax highlighting and Outline view
- autoconf often used to create Makefiles for open source projects
  - Must refresh after running configure script
- Refresh whenever file system is modified outside of Eclipse
Configuring Project Code

- Most projects will now have to be configured
  - This is project dependent
  - Do whatever is needed, e.g.
    - Run ./configure from a terminal window
    - Create external command to run configure
  - This should create/configure all project Makefiles
- Refresh the project to sync with file system
  - Right-click on project and select Refresh
Create a Make Target named ‘all’
- Right-click on the project in Make Targets View
- Select Add Make Target
- Select Create
- Double click on new make target to initiate the build
Create a Task Tag

- Task tags are identifiers in C/C++ comments
- TODO is a built-in task tag
- The build locates task tags during compilation
- View task tags in Tasks View
  - If it’s not shown, Window → Show View → Other...
  - Open General and select Tasks
- Configure your own task tag in Window → Preferences
  - Under C/C++, select Task Tags
Create a Bookmark

- A bookmark reminds you of useful information
- Add a bookmark by right-clicking in the gray border on left side of editor and select Add Bookmark…
  - Provide a bookmark name, then select OK
- View bookmarks by selecting Window ► Show View ► Other…
  - Open General and select Bookmarks
Commit Changes

- Select the **Project Explorer** view
- Notice the ‘>’ before the file name(s)
  - Indicates a file has been modified
- Right-click on the project
  - Select **Team ▶ Synchronize With Repository**
  - Confirm switch to perspective if asked
- Expand the project folder
  - Double-click on a file name to view differences
- Commit changes
  - Right-click on the file name, select **Commit...** and enter a comment
  - Select **Finish**
Refactoring

- **Rename**
  - Select **C/C++ Perspective**
  - Open a source file
  - Click in editor view on declaration of a variable
  - Select menu item **Refactor ▶ Rename**
    - Or use context menu
  - Change variable name
  - Notice that change is semantic not textual
Searching

- Language-based searching
- Search for Language Elements
  - e.g., C++ Class, Function, Method, Variable, Field, Namespace
- Limit search to Declarations, Definitions, References
- Type navigation
Type Navigation

- Choose **C/C++ Perspective**
- Select **Navigate ➤ Open Element**...
- Enter a name in text box
- All matching types are displayed
Module 5: PTP and Parallel Language Development Tools

Objective
- Learn to develop and run a parallel program

Contents
- Learn to use PTP’s Parallel Language Development Tools
- Learn to launch a parallel job and view it via the PTP Runtime Perspective
Parallel Tools Platform (PTP)

- The Parallel Tools Platform aims to provide a highly integrated environment specifically designed for parallel application development.
- Features include:
  - An integrated development environment (IDE) that supports a wide range of parallel architectures and runtime systems
  - A scalable parallel debugger
  - Parallel programming tools (MPI/OpenMP)
  - Support for the integration of parallel tools
  - An environment that simplifies the end-user interaction with parallel systems
- http://www.eclipse.org/ptp
Parallel Language Development Tools (1)

Features

- Analysis of C and C++ code to determine the location of MPI and OpenMP Artifacts (Fortran planned)
- "Artifact View" indicates locations of Artifacts found in source code
- Navigation to source code location of artifacts
- Content assist via `ctrl+space` ("completion")
- Hover help
- Reference information about the MPI and OpenMP calls via Dynamic Help
Parallel Language Development Tools (2)

- More PLDT features:
  - New project wizard automatically configures managed build projects for MPI & OpenMP
  - OpenMP problems view of common errors
  - OpenMP “show #pragma region” action
  - OpenMP “show concurrency” action
  - MPI Barrier analysis - detects potential deadlocks
To use the PTP Parallel Language Development Tools feature for MPI development, you need to:
- Specify the MPI include path
- Specify the MPI build command

Open Window > Preferences...
- Open the PTP item
- Open the Parallel Language Development Tools item
- Select MPI
- Select New... to add MPI include path

If running OpenMP, add its include file location here too (we will cover that later)
Turn Autobuild Off

- Because we assume you don’t have MPI installed on your local machine, turn off auto-build to avoid errors
  - Select Project ➤ Build Automatically
  - It should now not be checked.
Create a new MPI project
- **File ⎯ New ⎯ C Project**
- Name the project ‘MyHelloProject’
- Under Project types, under Executable, select “MPI Hello World C Project” and hit Next
- On **Basic Settings** page, fill in information for your new project (Author name etc.) and hit Next
On the MPI Project Settings wizard page, make sure Add MPI project settings to this project is checked.

Change default paths, etc. if necessary (they are probably OK)

Hit Finish*.

*If you instead hit Next, then on the Select Configurations page, you can alter Project settings. Hit Finish.
Changing the Project Properties Manually

If you wish to change the way your MPI program is built:

- Open the project properties
- Select C/C++ Build
- Select Settings
- Select GCC C Compiler
  - Change the command
- Select GCC C Linker
  - Change the command
- It’s also possible to change compiler/linker arguments
- The MPI Project wizard set these for you, so it isn’t necessary to change them.

Note: compiler/linker names will vary by platform.
Content Assist

- Open the C Editor by double-clicking on the "MyHelloProject.c" source file, which may be in the 'src' folder in your new MyHelloProject.

- Type an incomplete MPI function name e.g. "MPI_Ini" into the editor, and hit **ctrl-space**

- Select desired completion value with cursor or mouse

- Hover over the MPI Artifact identified in the source file to see additional information about that function call, for example
Context Sensitive Help

- Click mouse, then press help key when the cursor is within a function name
  - Windows: F1 key
  - Linux: ctrl-F1 key
  - MacOS X: Help key or Help ▶ Dynamic Help

- A help view appears (Related Topics) which shows additional information
- Click on the function name to see more information
- Move the help view within your Eclipse workbench, if you like, by dragging its title tab
Modify Project

- Enter the following before the MPI_Finalize:
  
  ```
  MPI_Barrier;
  ```

- Type ctrl-space

- Select an MPI_Barrier from the list

- After the “(“, enter MPI_COMM_WORLD (use ctrl-space to help type if you like)

- Resulting line:
  ```
  MPI_Barrier(MPI_COMM_WORLD);
  ```

- Save file

- Build is not necessary for this sample

---

Module 5  SC 2007  5-11
MPI Artifacts

- Select source file; Run analysis by clicking on drop-down menu next to the analysis button and selecting **Show MPI Artifacts**
- Markers indicate the location of artifacts in editor
- In **MPI Artifact View** sort by any column (click on col. heading)
- Navigate to source code line by double-clicking on the artifact
- Run the analysis on another file and its markers will be added to the view
- Remove markers via
MPI Barrier Analysis

Verify barrier synchronization in C/MPI programs

Interprocedural static analysis outputs:

- For verified programs, lists barrier statements that synchronize together (match)
- For synchronization errors, reports counter example that illustrates and explains the error.
MPI Barrier Analysis - views

**MPI Barriers view**
Simply lists the barriers
Like MPI Artifacts view, double-click to navigate to source code line (all 3 views)

**Barrier Matches view**
Groups barriers that match together in a barrier set – all processes must go through a barrier in the set to prevent a deadlock

**Barrier Errors view**
If there are errors, a counter-example shows paths with mismatched number of barriers
MPI Barrier Analysis - example

To run MPI Barrier Analysis:

1. Create a sample program in an MPI project that uses `barrier1.c` on the TutorialCD in ‘samples’ folder
2. Select the file (or Project) in the Projects view
3. Run Barrier Analysis via the menu
4. See the output views. No errors! Examine matching sets.
5. Comment out one of the barriers and rerun analysis. Note barrier mismatch and thus error.
6. Now put a barrier in a function call, to test interprocedural analysis features. The function call can even be in another source file.
7. Rerun analysis and view errors.
OpenMP Managed Build Project

Create a new OpenMP project

- **File ➤ New ➤ C Project**
- Name the project e.g. ‘MyOpenMPproject’
- Select OpenMP Hello
- Select **Next**, then fill in other info like MPI project

- If you haven’t set up OpenMP preferences e.g. include file location, you’ll be reminded
Setting OpenMP Special Build Options

- OpenMP typically requires special compiler options.
- Open the project properties
- Select C/C++ Build
- Select Settings
- Select C Compiler
  - In Miscellaneous, add option(s).

- This isn’t necessary for PLDT OpenMP analysis, only for building real executable OpenMP programs 😊
Show OpenMP Artifacts

- Select source file, folder, or project
- Run analysis
- See artifacts in OpenMP view
Show Pragma Region

- Run OpenMP analysis
- Right click on pragma in artifact view
- Select Show #pragma region
- See highlighted region in C editor
Show Concurrency

- Insert the following 

  ```
  #pragma...
  ```

- Save the file
- Re-run OpenMP analysis
- Select this statement
- Select the context menu on the highlighted statement, and click **Show concurrency**
- Other statements will be highlighted in yellow
- The yellow highlighted statements can execute concurrently to the selected statement
Show OpenMP Problems

- After “Show OpenMP artifacts” analysis:
- Select OpenMP problems view
- Sample file `openMPproblems.c` is on TutorialCD in ‘samples’ folder
Running a Parallel Application

- The **PTP Runtime** perspective is provided for monitoring and controlling applications
- Some terminology
  - **Resource manager** - Corresponds to an instance of a resource management system (e.g. a job scheduler). You can have multiple resource managers connected to different machines.
  - **Queue** - A queue of pending jobs
  - **Job** - A parallel application
  - **Machine** - A parallel computer system
  - **Node** - Some form of computational resource
  - **Process** - An execution unit (may be multiple threads of execution)
PTP Runtime Perspective

- Resource managers view
- Machines view
- Node details view
- Jobs view
Adding a Resource Manager

- Right-click in Resource Managers view and select **Add Resource Manager**

- Choose the **ORTE** Resource Manager Type

- Select **Next >**
Setting Remote System Address

- Select **RSE** as the Remote service provider
- Click **New...** to create a new location
- Enter IP address or host name of the remote machine
- Select **Finish**
- Select the proxy server location you just created if it is not visible in the dropdown
Setting Proxy Server

- Click **Browse** to select the proxy server executable
- Open **Root** twisty
- Enter your **User ID** and **Password** when asked
- Check **Save user ID** and **Save password**
- Click **OK**
- Now navigate to and select /usr/local/bin/ptp_orte_proxy
- Click **Yes** if you see this warning
Setting Local System Address

- This is the address that the proxy uses to connect to Eclipse
- Select your local machine’s IP address from the dropdown
- Enter it manually if it’s not visible
- Click **Finish**
Starting the Resource Manager

- Right click on new resource manager and select **Start resource manager**
- If everything is ok, you should see the resource manager change to **green**
- If something goes wrong, it will change to **red**
System Monitoring

- Machine status shown in **Machines** view
- Node status also shown in **Machines** view
- Hover over node to see node name
- Double-click on node to show attributes
Getting Program Source

- Switch to the **CVS Repository Exploring** perspective
- Open the repository you created in module 4
- Open **HEAD**
- Right click on **MyMPIProject** and select **Check Out** (not **Check Out As...**)
- Click **Yes** to confirm overwrite (*if* you already have a project with this name)
- Switch to the **C/C++** perspective to check the project is in your workspace
- Switch back to the **PTP Runtime** perspective
Create a Launch Configuration

- Open the run configuration dialog **Run** ➤ **Open Run Dialog**...
- Select **Parallel Application**
- Select the **New** button
In **Main** tab, select the resource manager you want to use to launch this job

Then click the **Browse** button to select the **Parallel Project**

Next, click the **Browse** button to find the **Application program** (executable) on the remote machine

- Open **My Home**
- Open **MyMPIProject**
- Select **MPIProgram**
- Click **OK**
Complete the Resources Tab

- Select **Resources** tab
- Enter the number of processes for this job
  - 4 is a good number for this tutorial
- Other resource managers may provide additional resources to select (e.g. network interface, run duration, etc.)
Complete the Debugger Tab

- Select **Debugger** tab
- Choose **SDM** from the **Debugger** dropdown
- Click on **Browse** and select the debugger executable
- For the tutorial:
  - Open **Root**
  - Navigate to `/usr/local/bin/sdm`
  - Click **OK**
- Click on the **Run** button to launch the job
Viewing The Run

- Double-click a node in machines view to see which processes ran on the node.
- Hover over a process for tooltip popup.
- Job and processes shown in jobs view.
Viewing Program Output

- Double-click a process to see process detail and standard output from the process.
About PTP Icons

- Open using legend icon in toolbar
Module 6: Parallel Debugging

Objective
- Learn the basics of debugging parallel programs with PTP

Contents
- Launching a parallel debug session
- The PTP Debug Perspective
- Controlling sets of processes
- Controlling individual processes
- Parallel Breakpoints
- Terminating processes
Launching A Debug Session

- Use the drop-down next to the debug button (bug icon) instead of run button
- Select the MyMPIProject to launch
- The debug launch will use the same number of processes that the normal launch used (edit the Debug Launch Configuration to change)
The PTP Debug Perspective (1)

- **Parallel Debug view** shows job and processes being debugged.
- **Debug view** shows threads and call stack for individual processes.
- **Source view** shows a current line marker for all processes.
The PTP Debug Perspective (2)

- **Breakpoints** view shows breakpoints that have been set (more on this later)
- **Variables** view shows the current values of variables for the currently selected process in the **Debug** view
- **Outline** view (from CDT) of source code
Process Sets (1)

- Traditional debuggers apply operations to a single process
- Parallel debugging operations apply to a single process or to arbitrary collections of processes
- A process set is a means of simultaneously referring to one or more processes
Process Sets (2)

- When a parallel debug session is first started, all processes are placed in a set, called the **Root** set
- Sets are always associated with a single job
- A job can have any number of process sets
- A set can contain from 1 to the number of processes in a job
Operations On Process Sets

- Debug operations on the **Parallel Debug view** toolbar always apply to the current set:
  - Resume, suspend, stop, step into, step over, step return
- The current process set is listed next to job name along with number of processes in the set
- The processes in process set are visible in right hand part of the view

Root set = all processes
Stepping All Processes

- Click on the **Step Over** button.
- Observe that all process icons change to green, then back to yellow.
- Notice that the current line marker has moved to the next source line.
- **Step Over** twice more, until your source window looks like this.
Managing Process Sets

The remaining icons in the toolbar of the Parallel Debug view allow you to create, modify, and delete process sets, and to change the current process set.
Creating A New Process Set

- Select the processes you want in the set by clicking and dragging, in this case, the last three.
- Click on the Create Set button.
- Enter a name for the set, in this case workers, and click OK.
- You will see the view change to display only the selected processes.
Stepping Using New Process Set

- With the **workers** set active, click the **Step Over** button.
- You will now see two current line markers, the first shows the position of process 0, the second shows the positions of processes 1-3 (the **workers**).
Stepping An Individual Process

- Switch back to the **Root** set by clicking on the **Change Set** button.
- The buttons in the **Debug view** are used to control and individual process, in this case process 0.
- Click the **Step Over** button.
- Notice that the first current line marker disappears (it has merged with the second current line marker.)
Process Registration

- Process set commands apply to groups of processes
- For finer control and more detailed information, a process can be registered and isolated in the **Debug view**
- Registered processes, including their stack traces and threads, appear in the **Debug view**
- Any number of processes can be registered, and processes can be registered or un-registered at any time
Registering A Process

- To register a process, double-click its process icon in the **Parallel Debug view** or select a number of processes and click on the **register** button.

- The process icon will be surrounded by a box and the process appears in the **Debug view**.

- To un-register a process, double-click on the process icon or select a number of processes and click on the **unregister** button.

Groups (sets) of processes

Individual (registered) processes
Current Line Marker

- The current line marker is used to show the current location of suspended processes
- In traditional programs, there is a single current line marker (the exception to this is multi-threaded programs)
- In parallel programs, there is a current line marker for every process
- The PTP debugger shows one current line marker for every group of processes at the same location
Colors And Markers

- The highlight color depends on the processes suspended at that line:
  - **Blue:** All registered process(es)
  - **Orange:** All unregistered process(es)
  - **Green:** Registered or unregistered process with no source line (e.g. suspended in a library routine)

- The marker depends on the type of process stopped at that location

- Hover over marker for more details about the processes suspend at that location
Breakpoints

- Apply only to processes in the particular set that is active in the Parallel Debug view when the breakpoint is created.
- Breakpoints are colored depending on the active process set and the set the breakpoint applies to:
  - **Green** indicates the breakpoint set is the same as the active set.
  - **Blue** indicates some processes in the breakpoint set are also in the active set (i.e. the process sets overlap)
  - **Yellow** indicates the breakpoint set is different from the active set (i.e. the process sets are disjoint)
- When the job completes, the breakpoints are automatically removed.

```c
else{
    print
    MPI_Final
}
```
Creating A Breakpoint

- Select the process set that the breakpoint should apply to, in this case, the workers set
- Double-click on the left edge of an editor window, at the line on which you want to set the breakpoint, or right click and use the Parallel Breakpoint ➤ Toggle Breakpoint context menu
- Set the breakpoint on the call to MPI_Send()
Hitting the Breakpoint

- Switch back to the **Root** process set
- Notice that the breakpoint color changes to indicate that the active set and the breakpoint set are different
- Click on the **Resume** button in the **Parallel Debug view**
- The three worker processes have hit the breakpoint, as indicated by the yellow process icons and the current line marker
- Process 0 is still running as its icon is green
More On Stepping

- The **Step** buttons are only enabled when all processes in the active set are **suspended** (yellow icon).
- In this case, process 0 is still running.

- Switch to the **workers** set.
- You will now see the **Step** buttons become enabled.
- Step a couple of times to see what happens.
Breakpoint Information

- Hover over breakpoint icon
  - Will show the sets this breakpoint applies to
- Select **Breakpoints** view
  - Will show all breakpoints in all projects

![Breakpoint Information Example](image-url)
Breakpoints View

- Use the menu in the breakpoints view to group breakpoints by type
- Breakpoints sorted by breakpoint set (process set)
Global Breakpoints

✦ Apply to all processes and all jobs
✦ Used for gaining control at debugger startup
✦ To create a global breakpoint
  ✦ First make sure that no jobs are selected (click in white part of jobs view if necessary)
  ✦ Double-click on the left edge of an editor window
  ✦ Note that if a job is selected, the breakpoint will apply to the current set

```c
if (my_rank != 0) {
    /* create message */
    sprintf(message, "Greetin
```
Terminating A Debug Session

- Click on the **Terminate** icon in the **Parallel Debug view** to terminate all processes in the active set.
- Make sure the **Root** set is active if you want to terminate all processes.
- You can also use the terminate icon in the **Debug** view to terminate the currently selected process.
Module 7: Where To Go Next

Objective

- How to find more information on PTP
- Learn about other tools related to PTP
- See PTP upcoming features

Contents

- Links to other tools, including performance tools
- Planned features for new versions of PTP
- Additional documentation
- How to get involved
Information About PTP

- Main web site for downloads, documentation, etc.
  - http://eclipse.org/ptp

- Developers wiki for designs, planning, meetings, etc.
  - http://wiki.eclipse.org/PTP

- Mailing lists
  - Major announcements (new releases, etc.) - low volume
    - http://dev.eclipse.org/mailman/listinfo/ptp-announce
  - User discussion and queries - medium volume
    - http://dev.eclipse.org/mailman/listinfo/ptp-user
  - Developer discussions - high volume
    - http://dev.eclipse.org/mailman/listinfo/ptp-dev
PTP-Related Tools

- Tuning and Analysis Utilities (TAU)
- TuningFork - Performance Visualization
- Remote System Explorer
Tuning and Analysis Utilities

Demo presented by Wyatt Spear, wspear@cs.uoregon.edu
http://www.cs.uoregon.edu/research/tau/

- **TAU Features**
  - Highly scalable and portable: works on numerous operating systems and architectures
  - Supports many data collection and analysis options, including hardware counters, callpath profiling and memory profiling
  - Allows output and conversion of performance data to several trace and profile formats

- **TAU Eclipse Plug-ins**
  - Simple configuration of TAU instrumentation and data collection options
  - Automatic 'one-click' instrumentation, compilation, execution and data-collection
  - Profile database and analysis tools integrated with Eclipse, including source callback
TuningFork

- Performance visualization Eclipse plug-ins from IBM Research
- Rich Client Platform or IDE versions available
- Designed for real-time visualization of large data sets
- Will be available open source on Source Forge
- Enhancements for parallel computing underway
Remote System Explorer

- http://eclipse.org/dsdp/tm
- Now part of the Eclipse releases (available in Europa)
- Allows project explorer to be used to view and manipulate remote files
- Supports connections using ssh, ftp, telnet, and its own protocol (dstore)
- Remote shell access
- List of remote processes
- Remote debugging (gdbserver)
- Not quite full remote project support
Useful Eclipse Tools

- Python

- Subversion (CVS replacement)
  - http://subclipse.tigris.org
  - Now an Eclipse Technology project

- Photran – Fortran Development Tools
  - http://eclipse.org/photran
  - Still under development

- ... and many more!
PTP Upcoming Features

- Additional resource manager support for MPICH2, SLURM, LoadLeveler, and Parallel Environment (PE)
- PLDT enhancements
  - Improved error checking for MPI and OpenMP
  - New static analysis, including parallelization assistance
- Full remote project support (combined with CDT)
  - Remote build and indexing
  - Remote launch/debug
PTP Upcoming Features (2)

- Debugger improvements
  - Support for new architectures
  - Scalability improvements
  - New user interface functionality
- Performance Analysis Framework
  - Provide integration for instrumentation, measurement, and analysis for a variety of performance tools
PTP Publications

  - Link on http://eclipse.org/ptp web page
  - Link on http://eclipse.org/ptp web page
- IBM developerWorks article:
  - http://www.isi.edu/~mhall/stmcs07/program.html
Getting Involved

- Read the developer documentation on the wiki
- Join the mailing lists
- Attend the monthly developer teleconference
- Attend the annual workshop

- PTP will only succeed with your participation!
PTP Tutorial Feedback

- Please complete feedback form
- Your feedback is valuable!

Thanks for attending
We hope you found it useful