Module 7: Advanced Development

- Objective
  - Become familiar with other tools that help parallel application development
- Contents
  - Parallel Language Development Tools: MPI, OpenMP, UPC
    - Overview of UPC tools
  - Performance Tuning and other external tools:
    - PTP External Tools Framework (ETFw), TAU
    - Parallel Performance Wizard (PPW)
  - MPI Analysis: GEM (Graphical Explorer of MPI Programs)

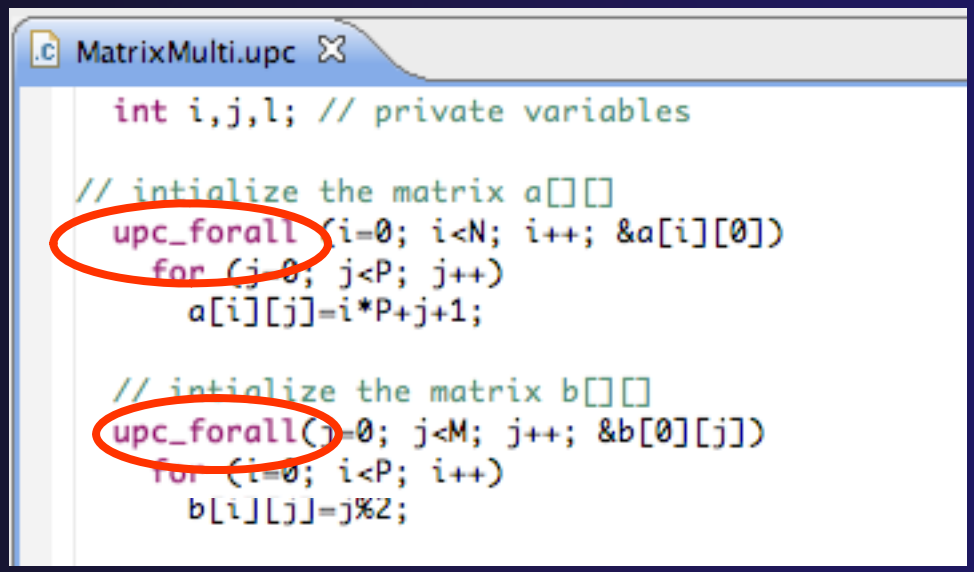
Eclipse UPC Features

- CDT:
  - Parser/Editor support
  - Code templates
  - IBM XLc (incl. xlUPC) – remote
  - Berkeley UPC toolchain – local (see backup slides)
- PTP:
  - Artifact identification; Hover/dynamic help assistance
  - More Code templates
  - Remote UPC parsing and builds with xlupc
  - Parallel Performance Wizard integration with PTP

Demo

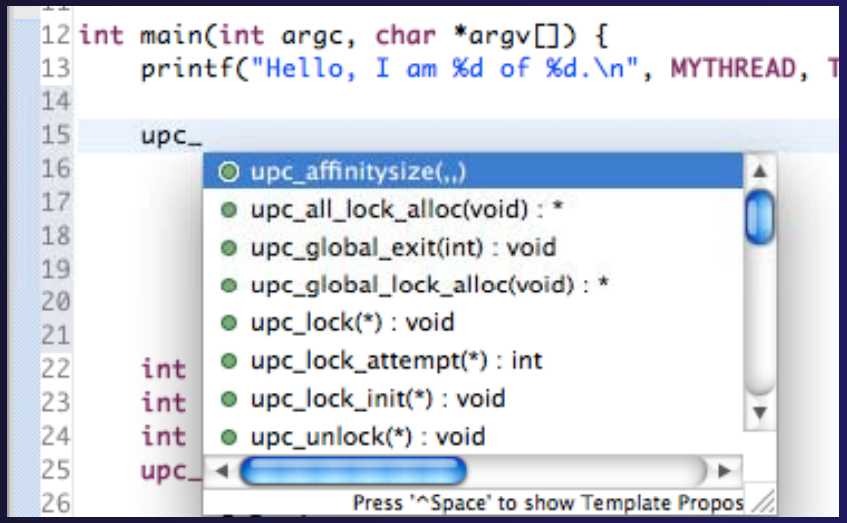
CDT - UPC Support

- Filetypes of “upc” will get UPC syntax high-lighting, content assist, etc.
- Use Preferences to change default for *.c if you like (we’ll show you how)

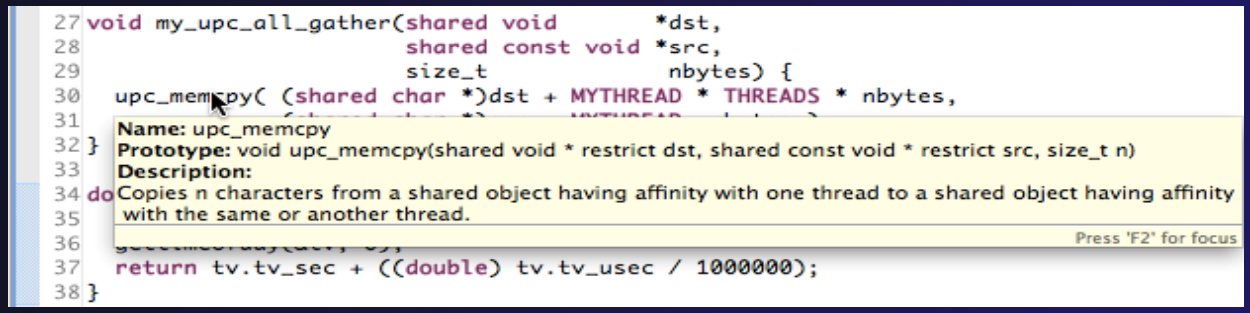


UPC Content Assist, Hover Help

- In Editor, type upc and hit control-space (once)
- A list of possible completions is provided.
- Choose with mouse or cursor.

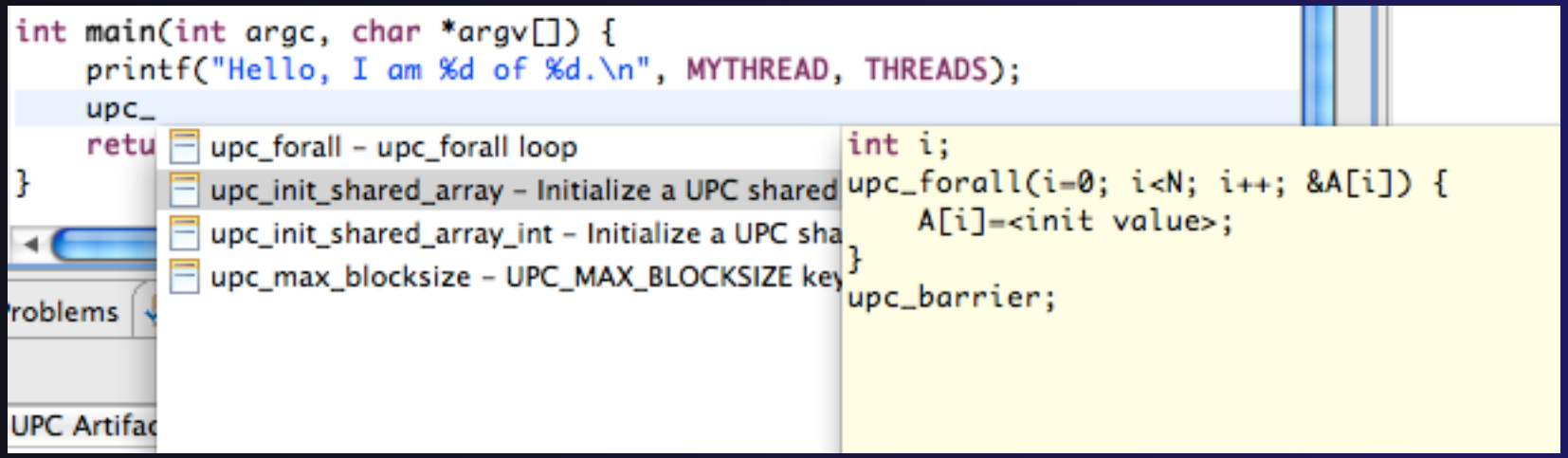


- Hover over API
- Hyperlink too



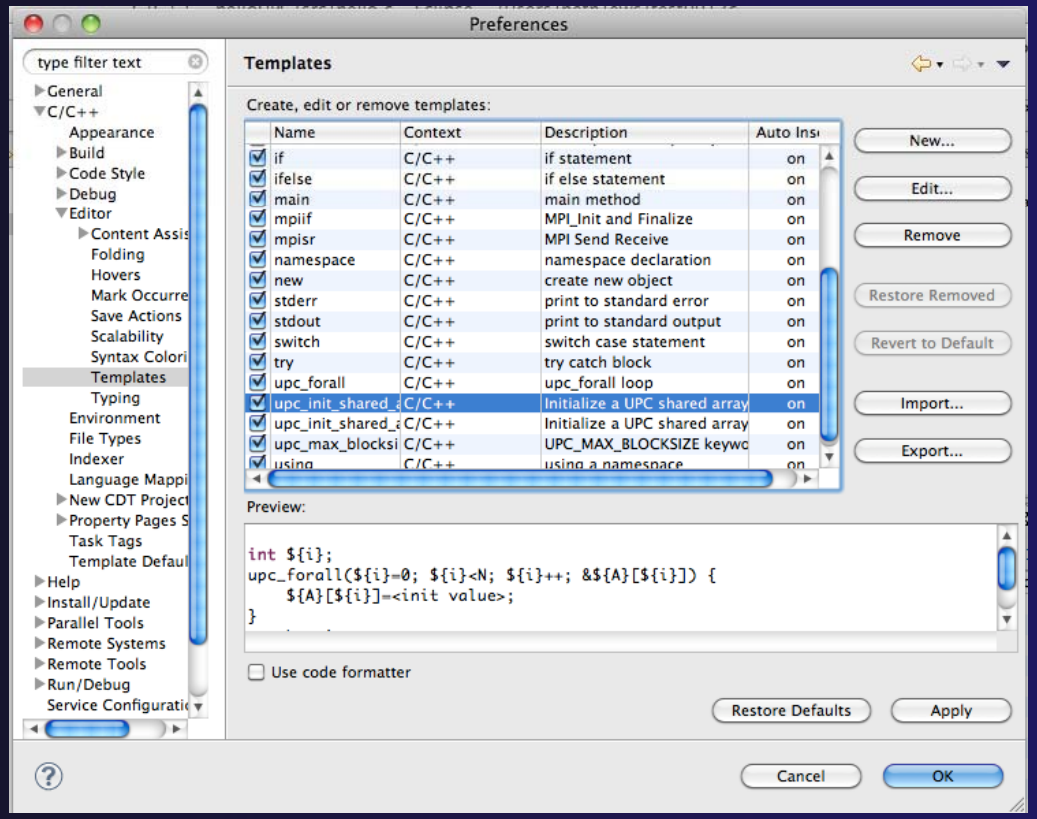
UPC templates - using

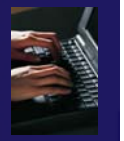
- In Editor, type
  upc and hit control-space
  (twice)



UPC templates – viewing/adding

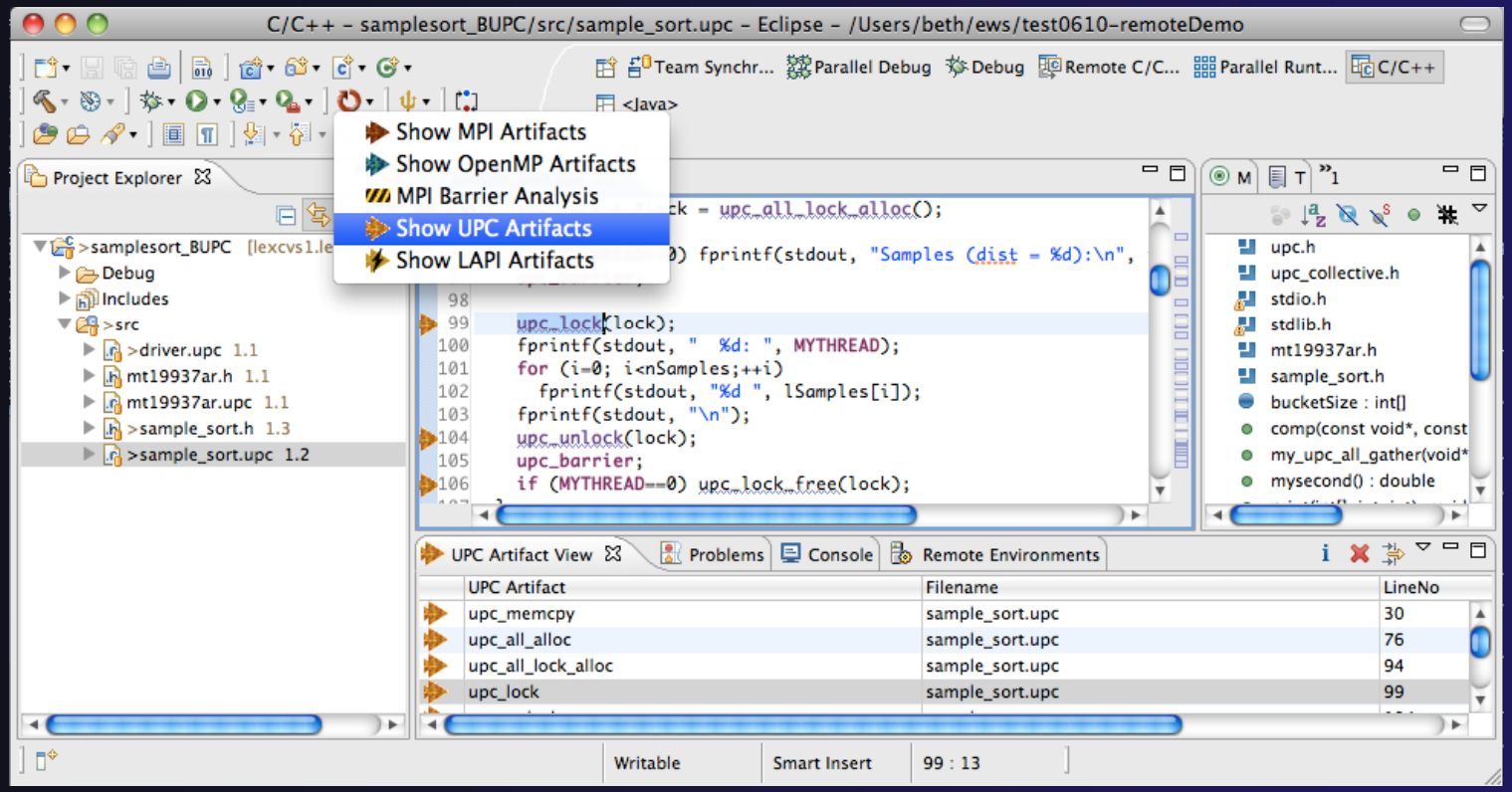
- Eclipse preferences: add more! Or just see what's there
  - **C/C++ > Editor > Templates**





Show UPC Artifacts

- Add some UPC api's to your sample project
- Show UPC Artifacts



Other UPC features

- UPC parser is remote-enabled
  - Remote UPC projects can be developed efficiently
- Remote xlUPC toolchain enables remote build of IBM xlUPC project
  - Managed Build (user-friendly) way to specify and manage complex build options without makefiles

More Advanced Features: Demos

- ETFw – External Tools Framework and TAU, Tuning and Analysis Utilities
  - Wyatt Spear, U. Oregon
- PPW – Parallel Performance Wizard
  - Max Billingsley III, U. Florida
- GEM – Graphical Explorer of MPI Programs
  Dynamic Formal Verification for MPI
  - Alan Humphrey, U. Utah

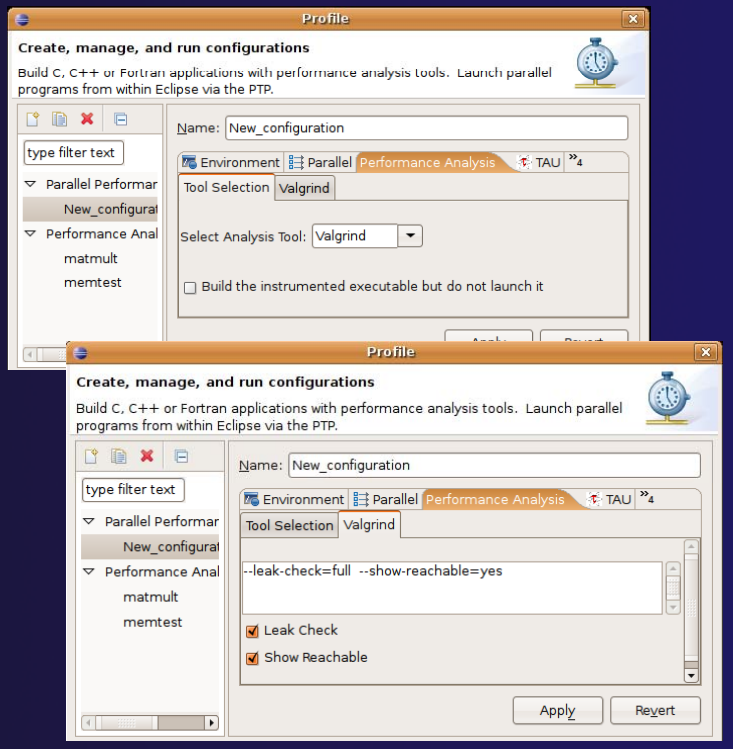
PTP/External Tools Framework

formerly “Performance Tools Framework”

Goal:

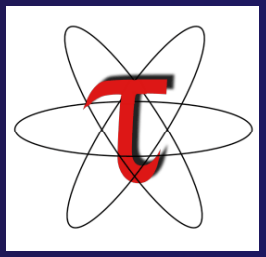
- **Reduce the “eclipse plumbing” necessary to integrate tools**
- Provide integration for instrumentation, measurement, and analysis for a variety of performance tools
  - Dynamic Tool Definitions: Workflows & UI
  - Tools and tool workflows are specified in an XML file
  - Tools are selected and configured in the launch configuration window
  - Output is generated, managed and analyzed as specified in the workflow

```
−<tool name="Valgrind">
 −<execute>
    <utility command="bash" group="inbin"/>
   −<utility command="valgrind" group="valgrind">
    −<optionpane title="Valgrind" seperatewith=" ">
       <togoption label="Leak Check" optname="--leak-check=full" tooltip
       <togoption label="Show Reachable" optname="--show-reachable=y
     </optionpane>
   </utility>
 </execute>
</tool>
```

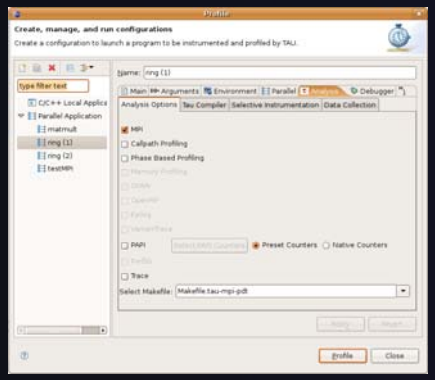


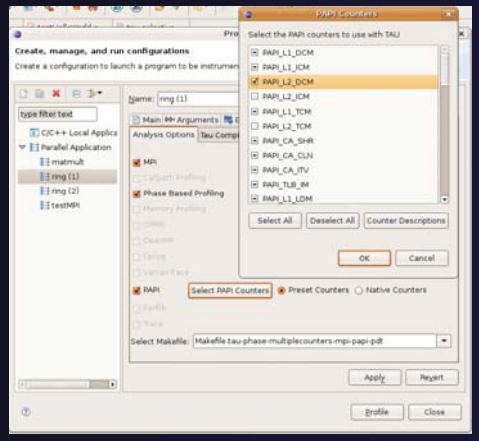
PTP TAU plug-ins

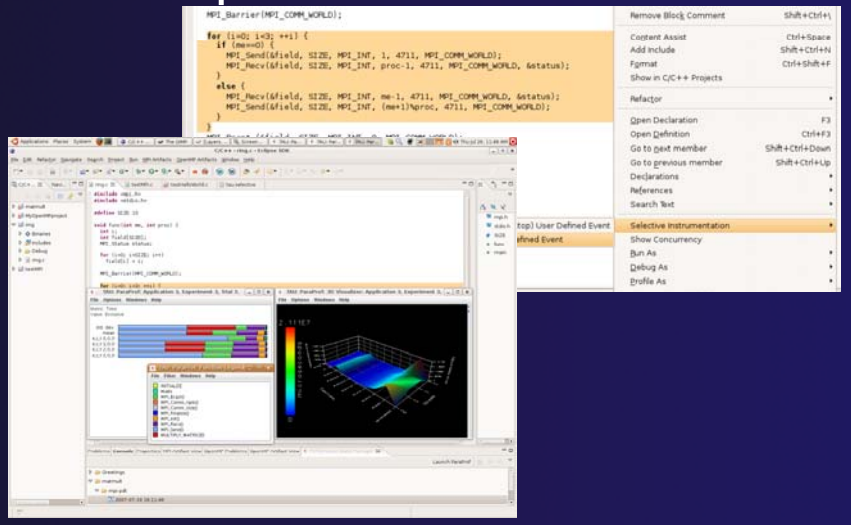
http://www.cs.uoregon.edu/research/tau



- TAU (Tuning and Analysis Utilities)
- First implementation of External Tools Framework (ETFw)
- Eclipse plug-ins wrap TAU functions, make them available from Eclipse
- Compatible with Photran and CDT projects and with PTP parallel application launching
- Other plug-ins launch Paraprof from Eclipse too

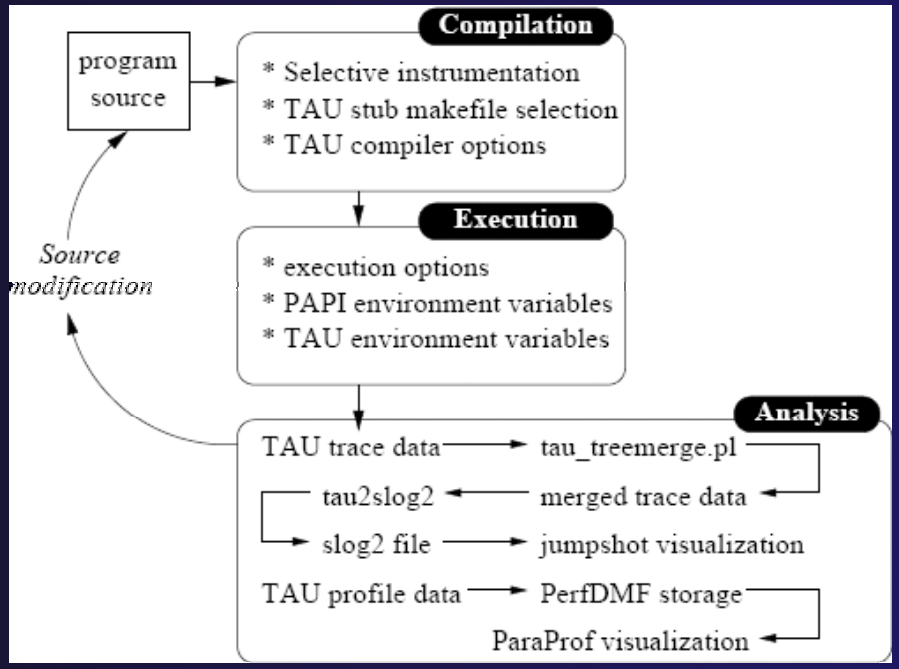






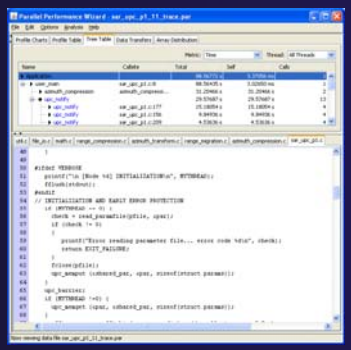
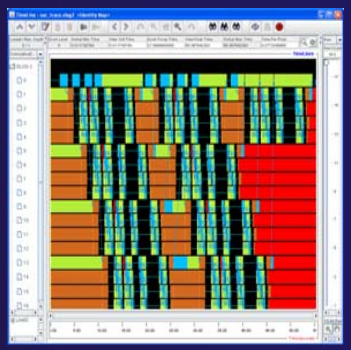
TAU Integration with PTP

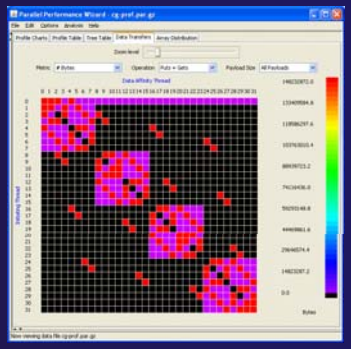
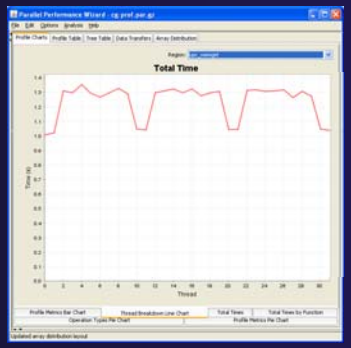
- TAU: Tuning and Analysis Utilities
  - Performance data collection and analysis for HPC codes
  - Numerous features
  - Command line interface
- The TAU Workflow:
  - Instrumentation
  - Execution
  - Analysis

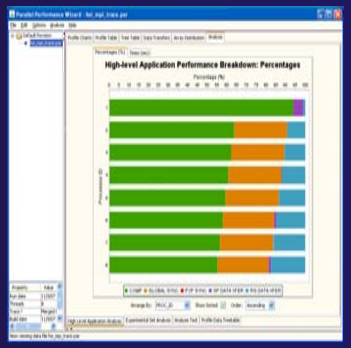


Parallel Performance Wizard (PPW)

- Full-featured performance tool for PGAS programming models
  - Currently supports UPC, SHMEM, and MPI
  - Extensible to support other models
  - PGAS support by way of Global Address Space Performance (GASP) interface (http://gasp.hcs.ufl.edu)
- PPW features:
  - Easy-to-use scripts for backend data collection
  - User-friendly GUI with familiar visualizations
  - Advanced automatic analysis support
- More information and free download: http://ppw.hcs.ufl.edu

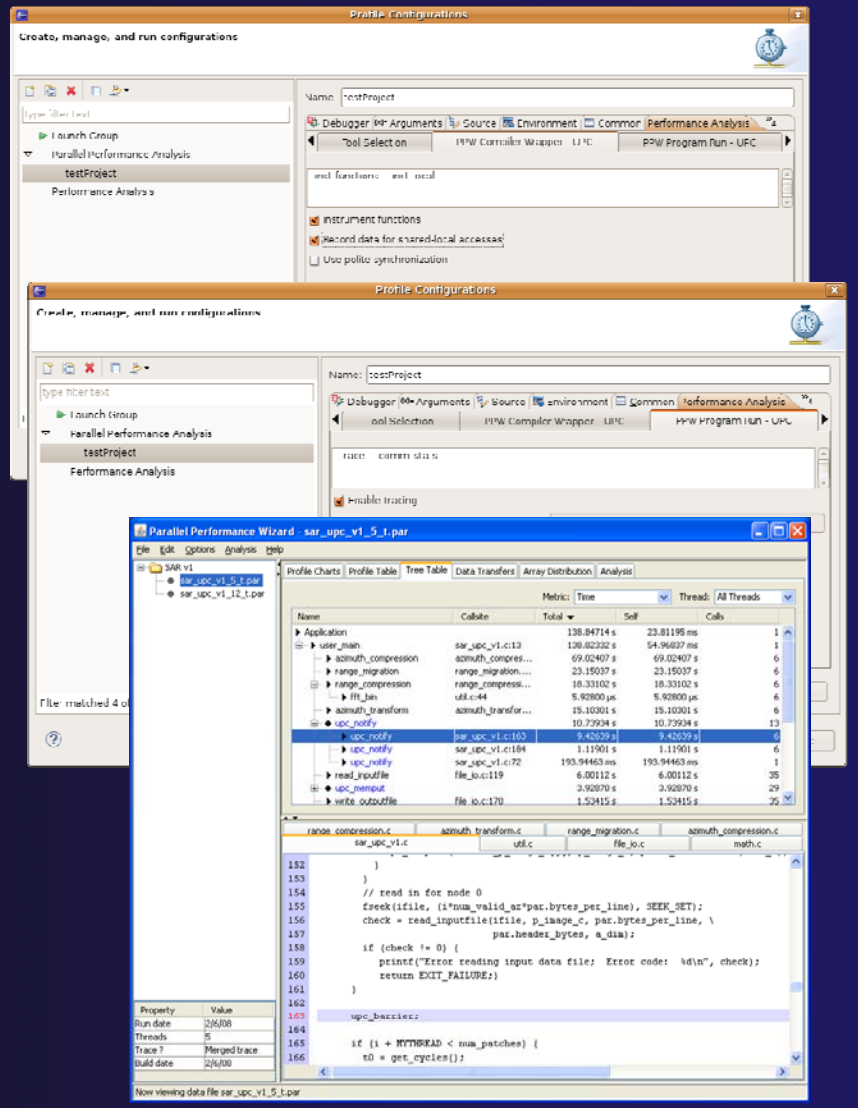





PPW Integration via ETFw

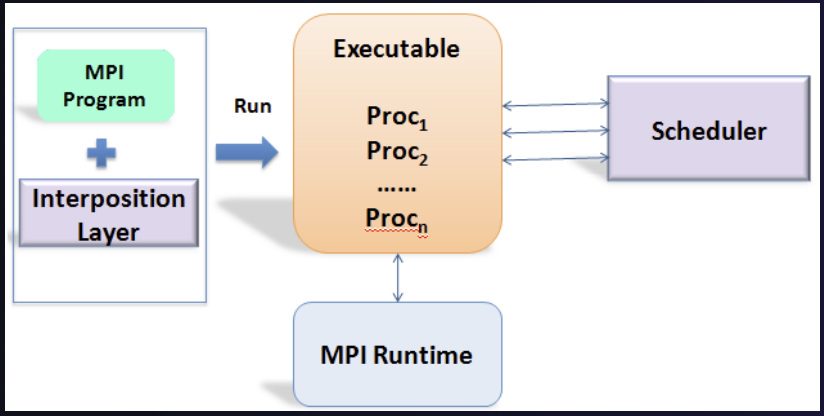
- We implement the ETFw to make PPW's capabilities available within Eclipse
  - Compile with instrumentation, parallel launch with PPW
  - Generates performance data file in workspace, PPW GUI launched
- PPW is often used for UPC application analysis
  - ETFw extended to support UPC
  - Many UPC features in PTP
- For more information:
  - http://ppw.hcs.ufl.edu
  - ppw@hcs.ufl.edu



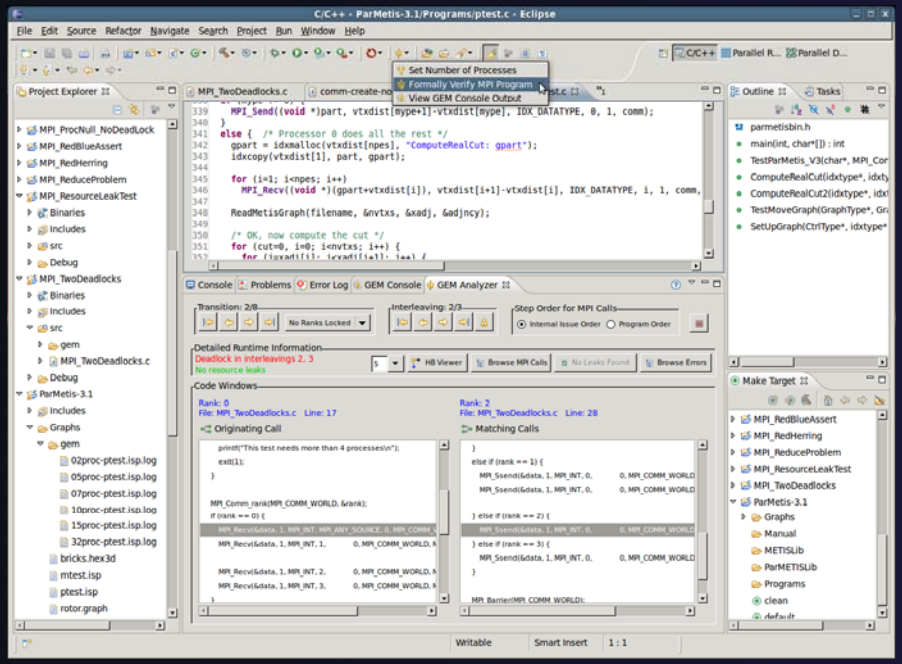
GEM - Graphical Explorer of MPI Programs

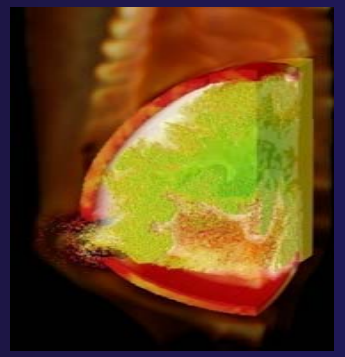
- Contributed to PTP by the University of Utah
  - Available with PTP since v3.0
- Dynamic verification for MPI C/C++ that detects:
  - Deadlocks
  - Local assertion violations
  - MPI object leaks
  - Functionally irrelevant barriers
- Offers rigorous coverage guarantees
  - Complete nondeterministic coverage for MPI
  - Communication / synchronization behaviors
  - Determines relevant interleavings, replaying as necessary

GEM - Overview



- Front-end for In-situ Partial Order (ISP), Developed at U. Utah
- Offers “push-button” verification from within the Eclipse IDE
- Automatically instruments and runs user code, displaying post verification results
- Variety of views & tools to facilitate debugging and code understanding





(Image courtesy of Steve Parker, U of Utah)

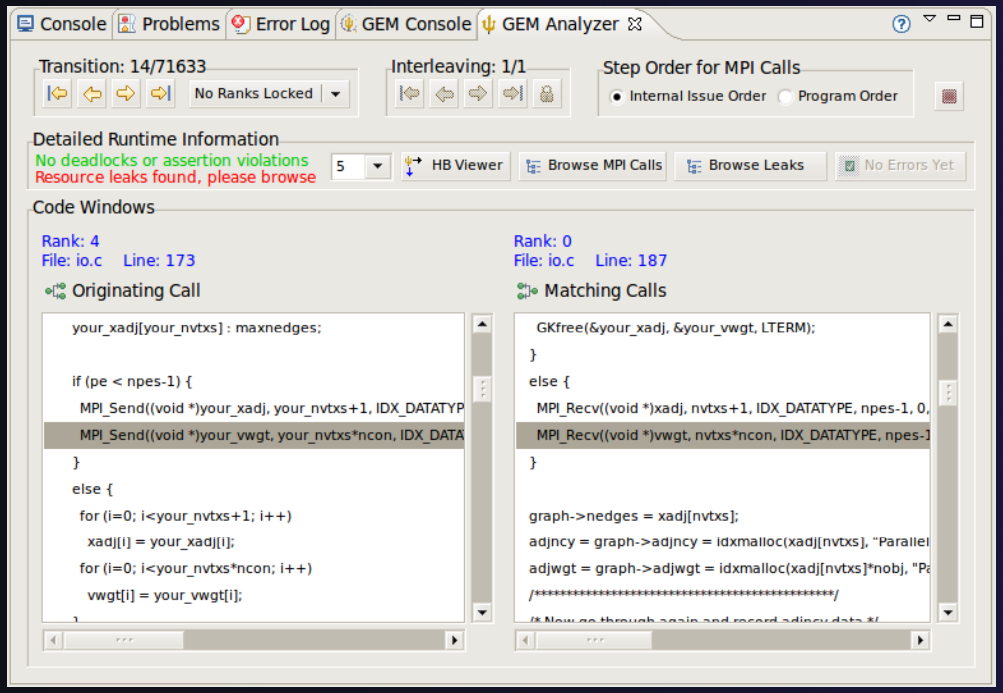
GEM – Views & Tools

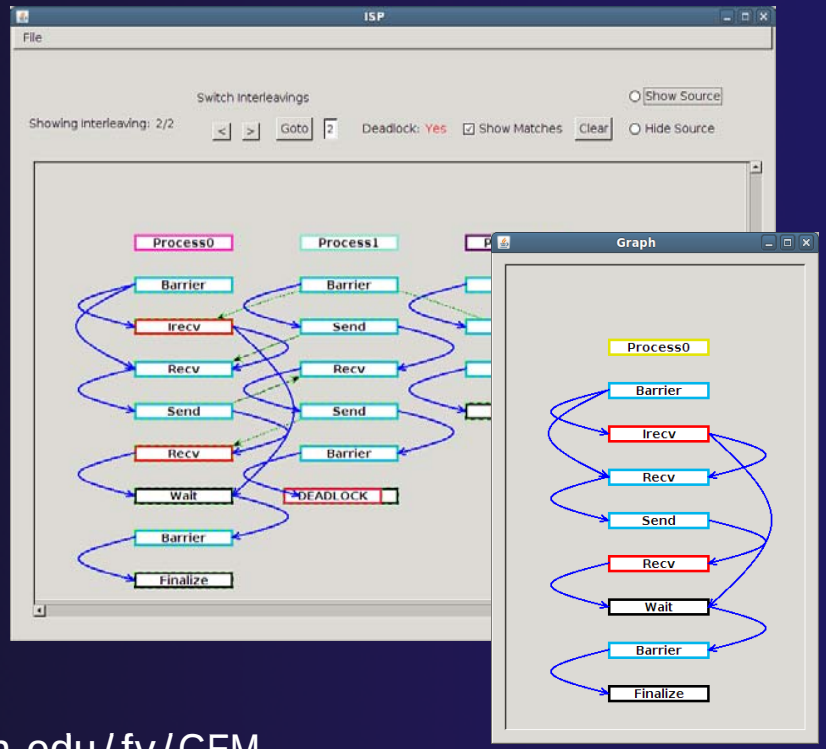
Analyzer View

Highlights Bugs, and facilitates Post-Verification Review / Debugging

Happens-Before Viewer

Shows required orderings and communication matches





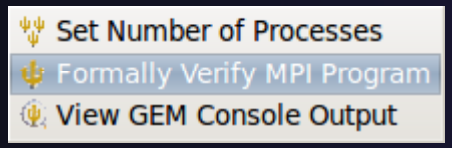
Download / documentation: http://www.cs.utah.edu/fv/GEM

Using GEM – ISP Installation

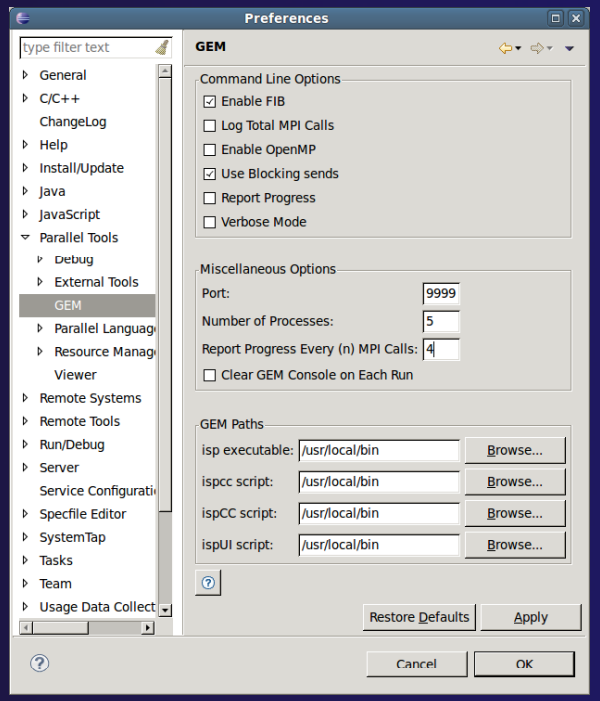
- **ISP itself must be installed prior to using GEM**
  - Download ISP at http://www.cs.utah.edu/fv/ISP
- Make sure libtool, automake and autoconf are installed.
- Just untar isp-0.2.0.tar.gz into a tmp directory and:
  - Execute the following commands from tmp directory
    - ./configure
    - make
    - make install
      - Do this with root privelage, sudo, etc. Puts binaries and necessary scripts in /usr/local/bin, /usr/local/lib, etc

Using GEM

- Create an MPI C Project within C/C++ Perspective
  - Make sure your project builds correctly
- Set preferences via GEM Preference Page
- From the trident icon or context menus user can:

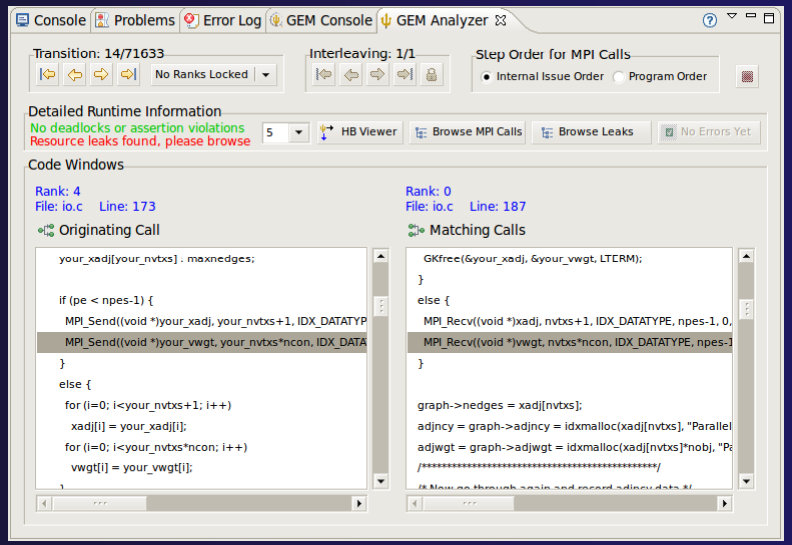


- Formally Verifying MPI Program
  - Launches ISP
  - Generates log file for post-verification analysis views
  - Opens relevant GEM views



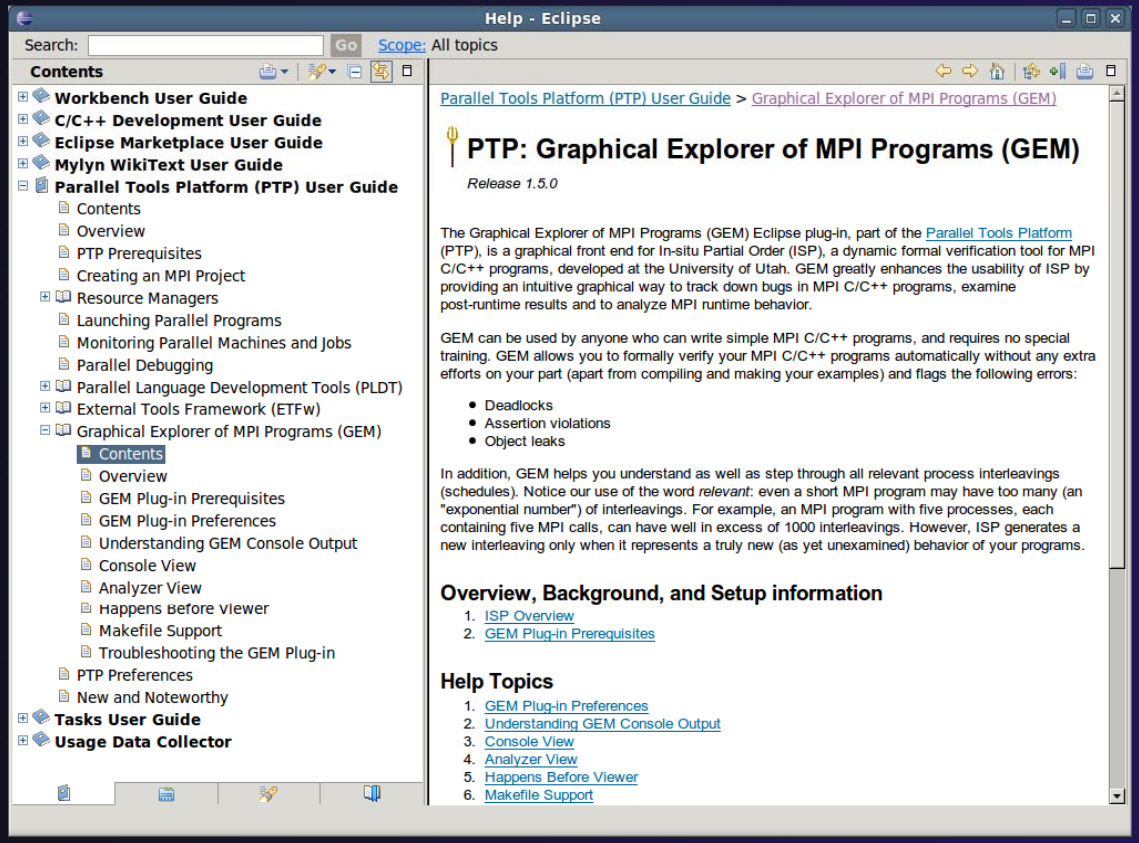
GEM Analyzer View

- Reports program errors, and runtime statistics
- Debug-style source code stepping of interleavings
  - Point-to-point / Collective Operation matches
  - Internal Issue Order / Program Order views
  - Rank Lock feature – focus in on a particular process
- One click to visit the Eclipse editor, to examine:
  - Calls involved in deadlock
    - Helps find root-cause
  - MPI Object Leaks sites
    - Locates allocated object
  - Local Assertion Violations
    - Takes user to failing assertion



GEM – Help Plugin

Extensive how-to sections, graphical aids and trouble shooting section

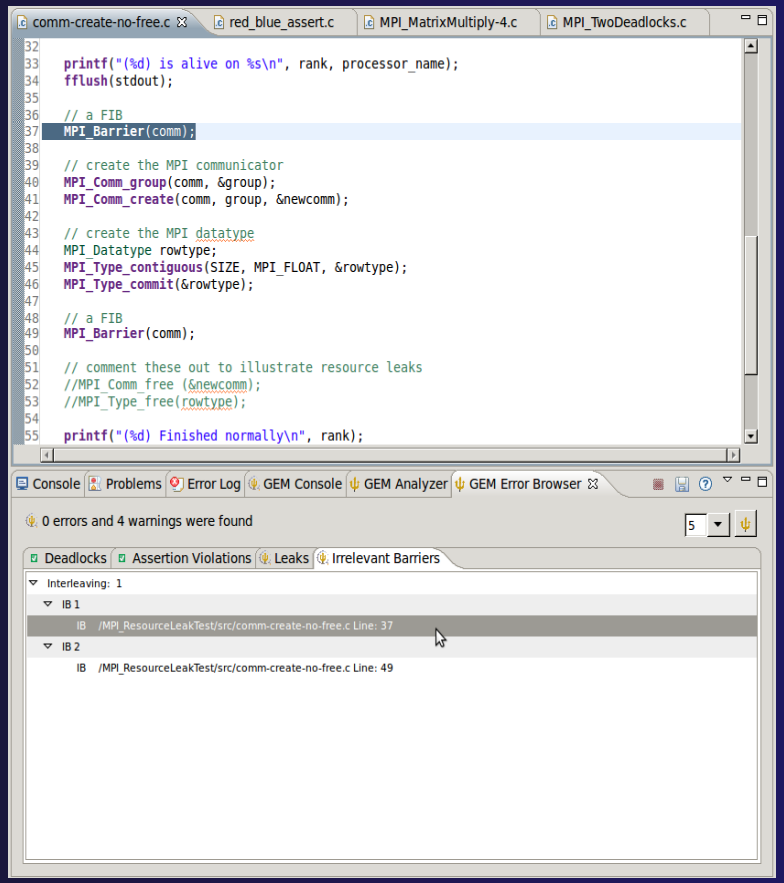


GEM/ISP Success Stories

- Umpire Tests
  - http://www.cs.utah.edu/fv/ISP-Tests
  - Documents bugs missed by tests, caught by ISP
- MADRE (EuroPVM/MPI 2007)
  - Previously documented deadlock detected
- N-Body Simulation Code
  - Previously unknown resource leak caught during EuroPVM/MPI 2009 tutorial !
- Large Case Studies
  - ParMETIS, MPI-BLAST, IRS (Sequoia Benchmark), and a few SPEC-MPI benchmarks could be handled
- Full Tutorial including LiveDVD ISO available
  - Visit http://www.cs.utah.edu/fv/GEM

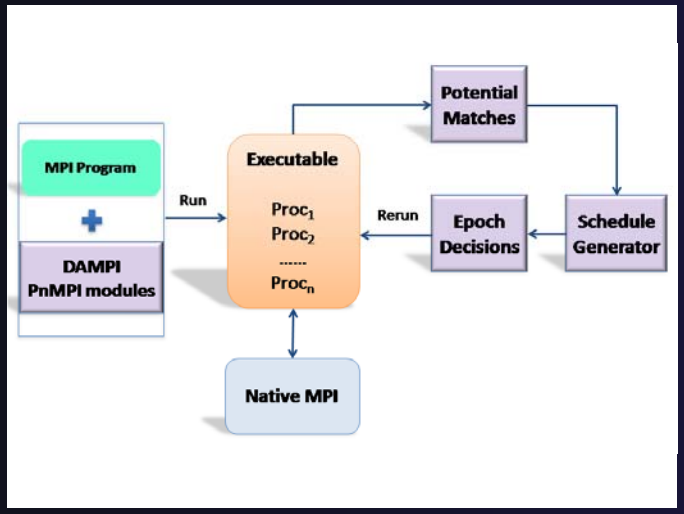
GEM Future Plans

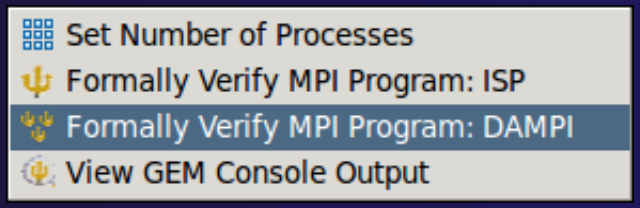
- Tabbed browsing for each type of error
- Each error mapped to offending line of source code in Eclipse editor
- Adding more error and property checks, e.g.
  - MPI send/recv type mismatch
  - Insufficient recv buffer
  - MPI argument mismatch
  - List unfreed requests at finalize



GEM Future Plans

- GEM will serve as a front-end for other tools
- Integration of Distributed Analyzer of MPI Programs (DAMPI), developed at University of Utah
  - ISP scales to 10s of processes
  - DAMPI scales to 1000s of processes (C/C++/Fortran)
  - Decentralized scheduler uses Lamport Clocks





Use **ISP** at small scale, then launch **DAMPI** at scale on a cluster

PTP Adv. Development: Summary

- A diversity of other tools aid parallel development
  - Parallel Language Development Tools:
    MPI, OpenMP, UPC, LAPI, etc.
  - External Tools Framework (ETFw) eases integration of existing (command-line, etc.) tools
    - TAU Performance Tuning uses ETFw
    - PPW (Parallel Perf. Wizard) uses ETFw for UPC analysis
    - Feedback view maps tool findings with source code
  - MPI Analysis: GEM
- A diversity of contributors too!
  - We welcome other contributions. Let us help!

Backup

- Not covered in today's tutorial, but included for reference
  - Creating a local MPI project, and using the wizards
    - MPI Assistance tools
    - MPI Barrier analysis on a local project
  - OpenMP tools
  - UPC tools installation and local projects
  - External Tools Framework (ETFw) details, overview of integrating other tools into PTP
  - ETFw Feedback view incl. sample exercise

Parallel Lang. Dev. Tools

- PLDT Features
  - Analysis of C and C++ code to determine the location of MPI, OpenMP, and UPC Artifacts
  - Content assist via **ctrl+space** (“completion”)
  - Hover help
  - Reference information about the API calls via Dynamic Help
  - New project wizard automatically configures managed build projects for MPI & OpenMP
  - OpenMP problems view of common errors
  - OpenMP “show #pragma region” , “show concurrency”
  - MPI Barrier analysis - detects potential deadlocks

Some MPI features were covered in Module 4

Note: Some PLDT features don’t work on remote (RDT) projects

MPI Assistance Tools

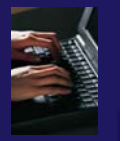
Added by PLDT (Parallel Lang. Dev. Tools) feature of PTP

- MPI Context sensitive help
- MPI artifact locations
- MPI barrier analysis
- MPI templates

- For this part, we will use the *local* MPI New Project Wizard and the “MPI Hello World” project

Creating Local Project

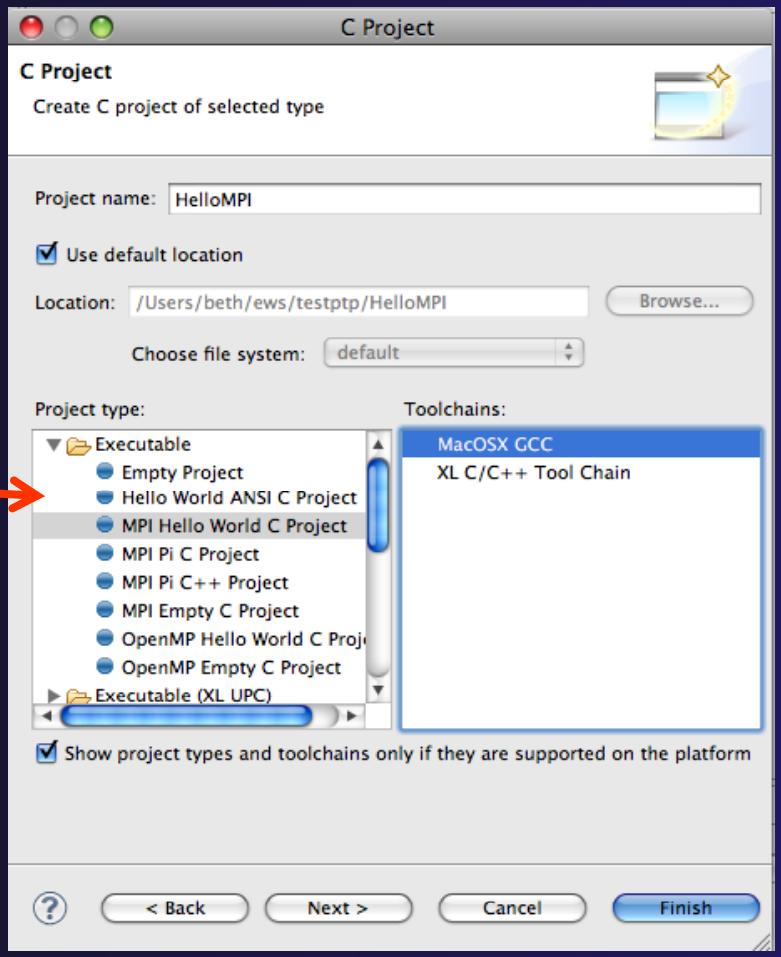
- The next slide shows you how to create a local MPI project.
- If you do not have MPI on your local machine, you can't build or run.
- *But* you should be able to demonstrate the MPI features in PTP's PLDT regardless.
- Several PLDT MPI features pertain to developing code – just using the local editor, etc.
- Most PLDT features *do* work on remote projects.

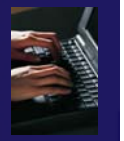


Create local MPI Project

Using a Managed Build Project – for a quick sample *local* MPI project

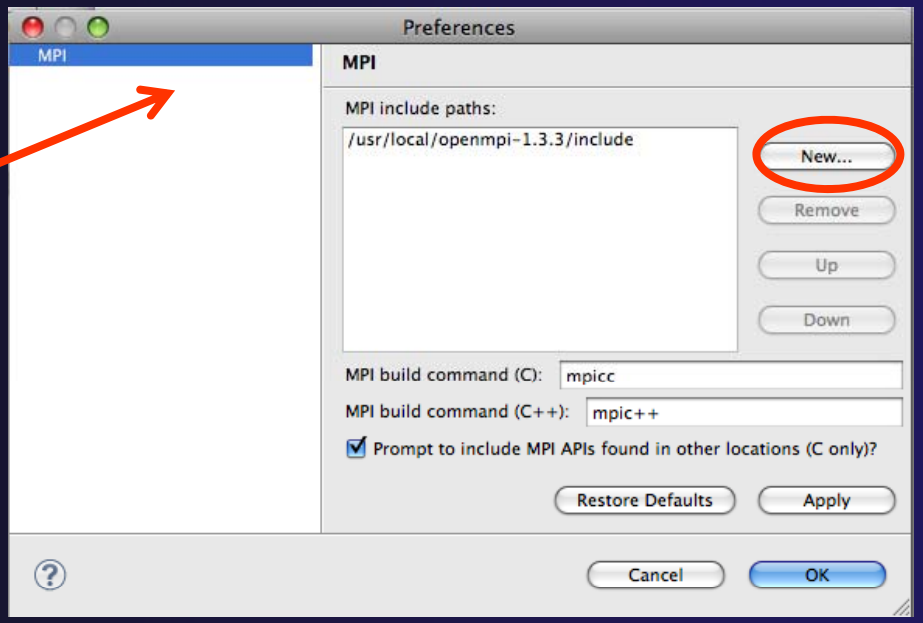
- **File > New > C Project**
- Give Project a name, e.g. HelloMPI
- Confirm Toolchain
- Select **MPI Hello World C Project**





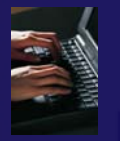
Set MPI Preferences

- When creating a local MPI project with the wizard, you need to set MPI Preferences (once)
- This assures the include paths, etc. will be set for new MPI projects – for building, and for Eclipse assistance features for MPI.
- Select **Yes** to set the MPI preferences.



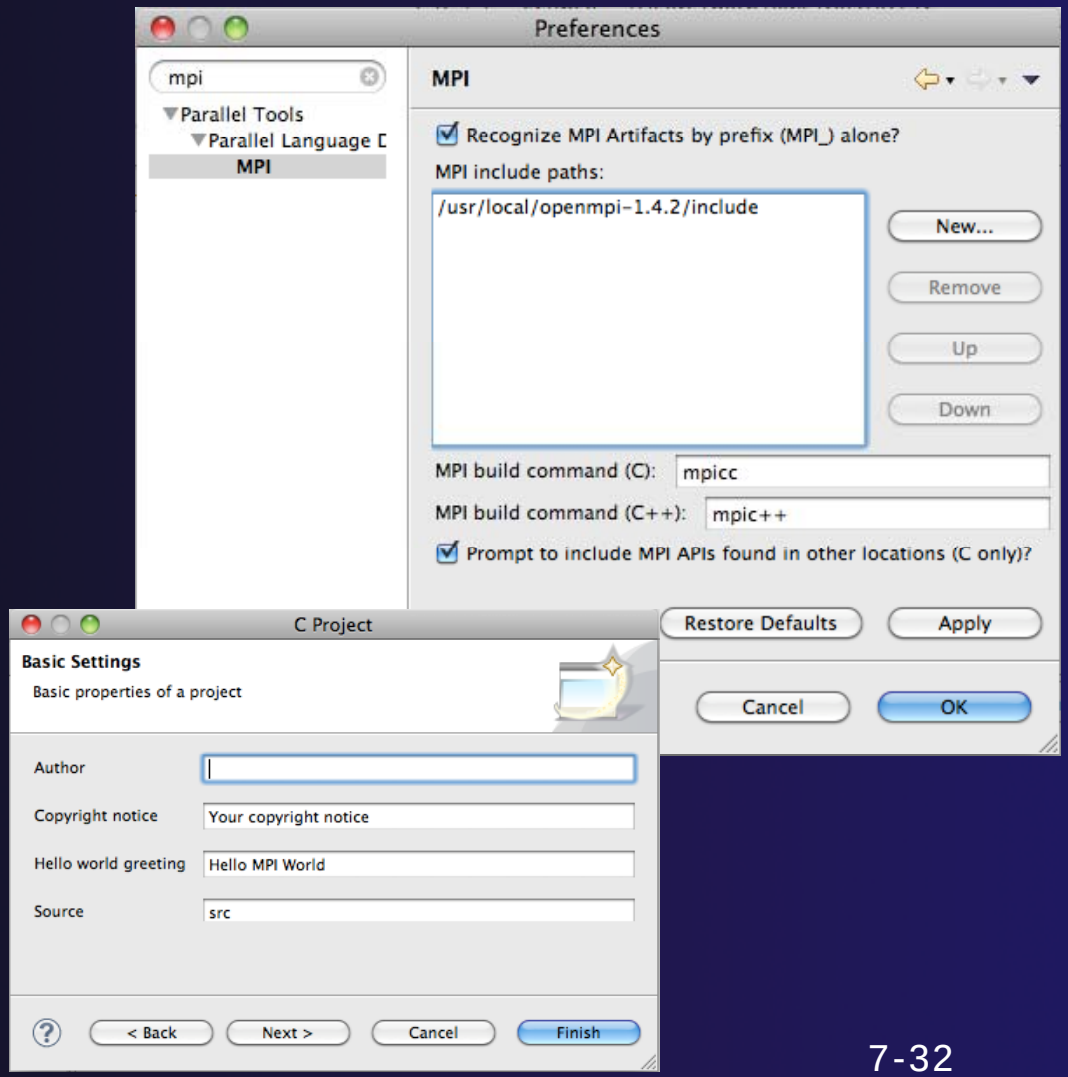
No MPI?

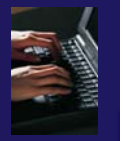
Note: if you do not have MPI on your local machine, you can use just an MPI header file (mpi.h) so you play with the PTP MPI development features without building or running on your local machine.



Set MPI Preferences (2)

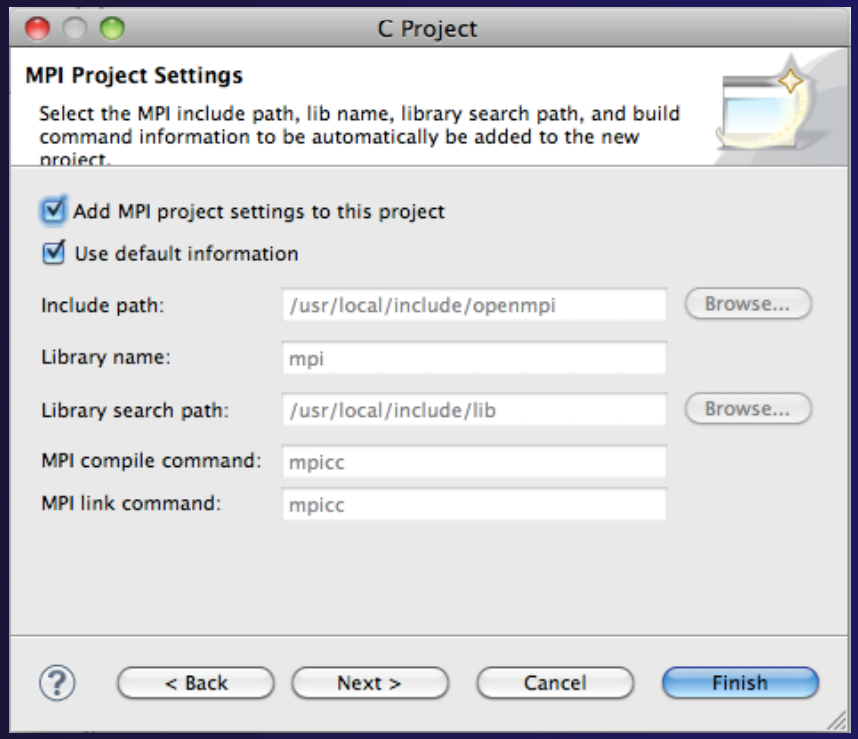
- On the MPI Preferences page, add a new MPI include path.
- New … and point to the *directory* containing your MPI header file (mpi.h)
- Select **OK**
- Back on New Project Wizard page, select **Next>** and fill in Author name, etc.

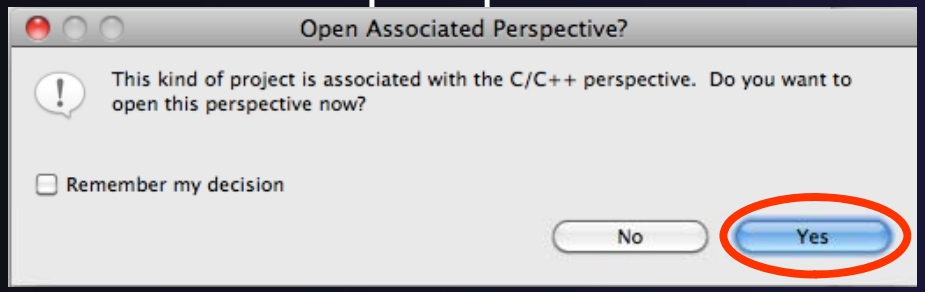


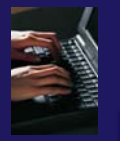


Review MPI Project Settings

- On the next wizard page, review the MPI project settings based on the information you have provided.
- Make changes if you wish.
- The defaults should be fine.
- Click **Finish.**
- You will be prompted to switch perspectives





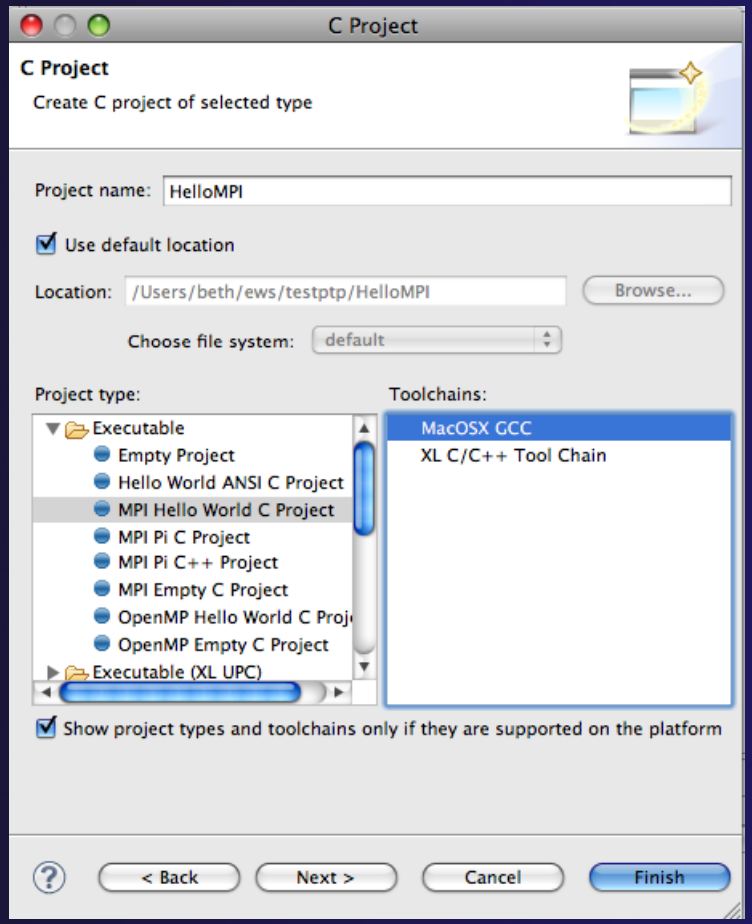


Create MPI Project

Recap:

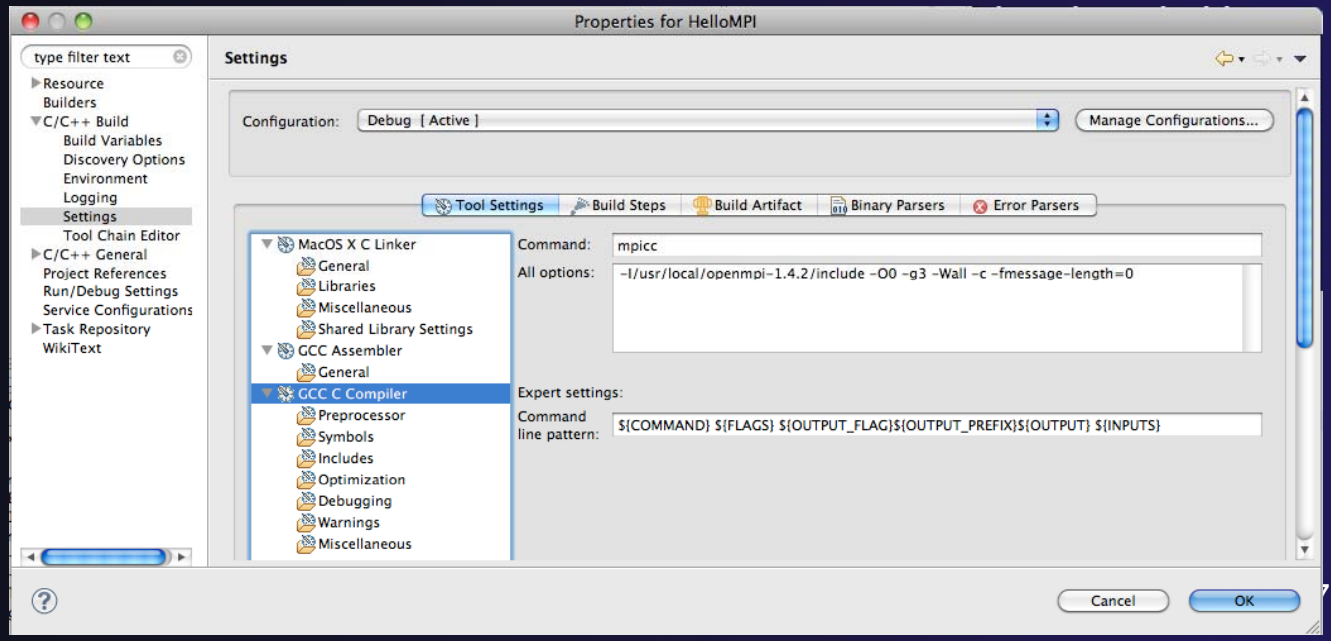
- File > New > C Project
- Give Project a name, e.g. HelloMPI
- Select Toolchain
- Select MPI Hello World C Project
- Set MPI Prefs, if first time
- Click Finish

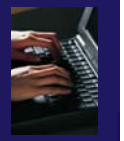
- Note: if it doesn't build on your machine, you can still continue with this exercise



Project Properties: Managed Build Project

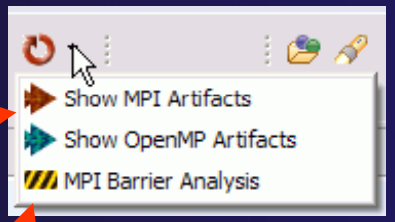
- Right-click on project in Project Explorer view and select **Properties**
- Project Properties for Managed Build project
  - Compiler, Linker, etc. settings set automatically without a Makefile



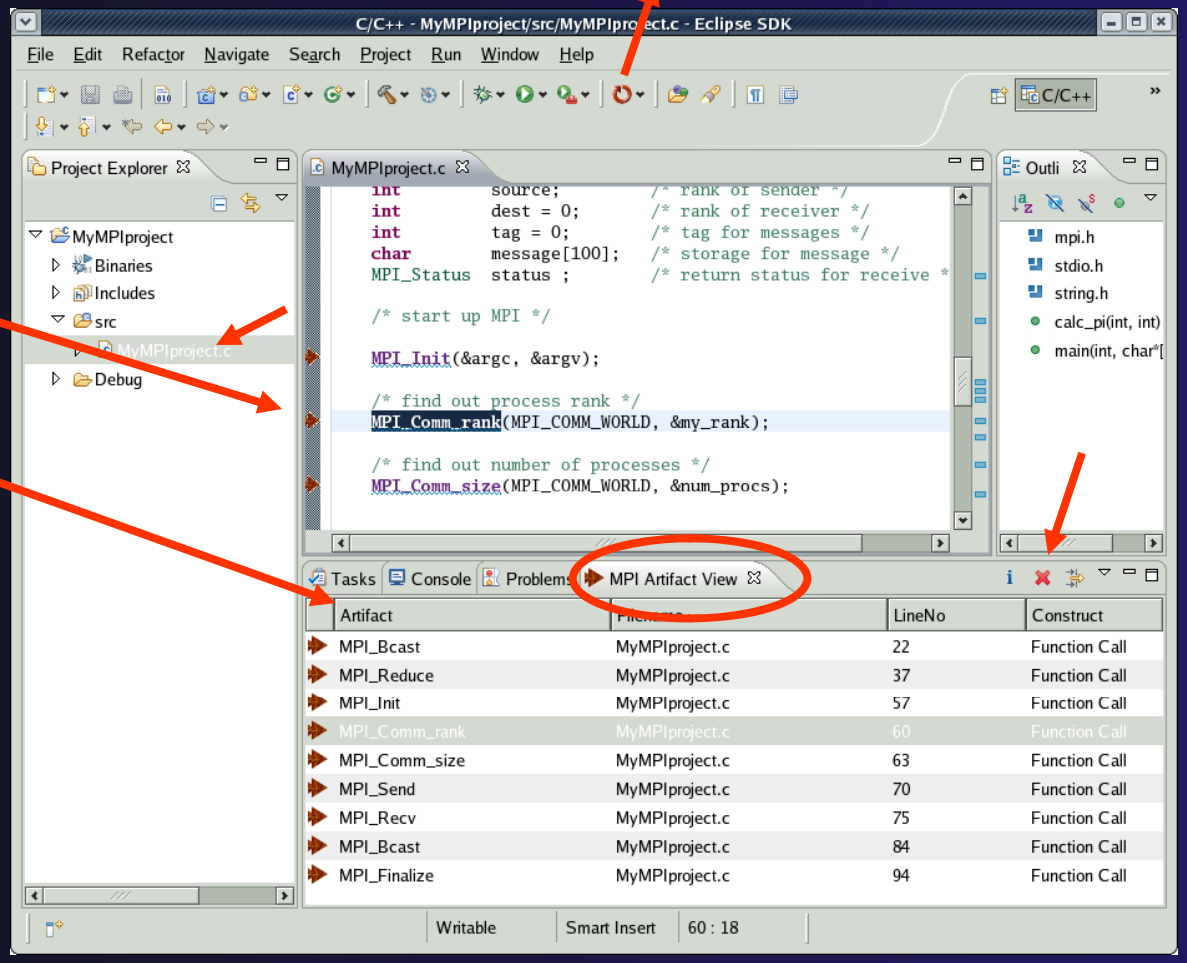


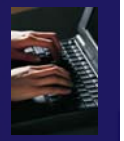
Show MPI Artifacts

- Select source file in Project Explorer; Select **Show MPI Artifacts** in PLDT menu



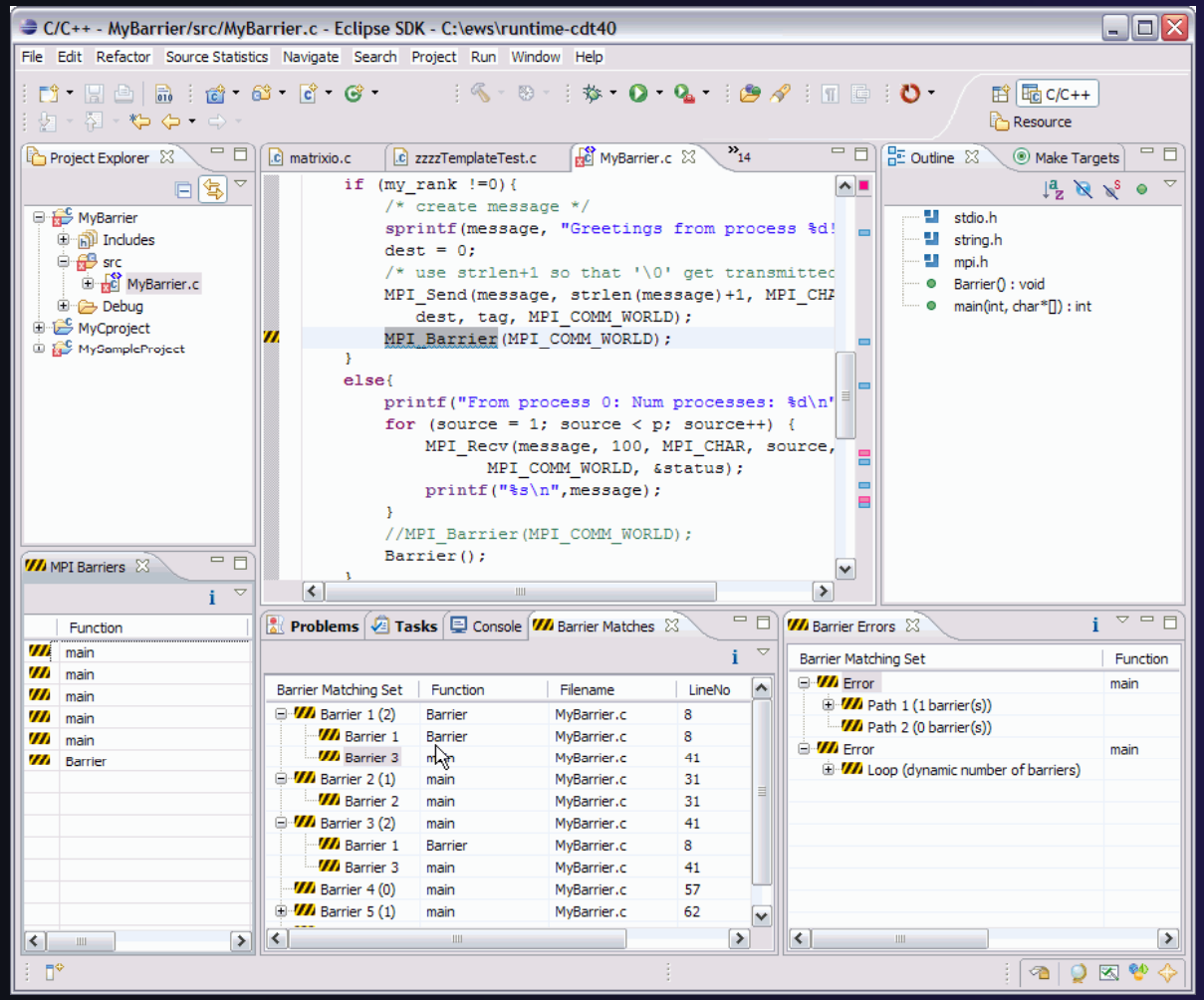
- Markers indicate the location of artifacts in editor
- In **MPI Artifact View** sort by any column (click on col. heading)
- Navigate to source code line by double-clicking on the artifact
- Run the analysis on another file and its markers will be added to the view
- Remove markers via 





MPI Barrier Analysis

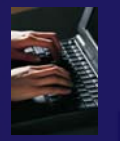
Local files only



Verify barrier synchronization in C/MPI programs

Interprocedural static analysis outputs:

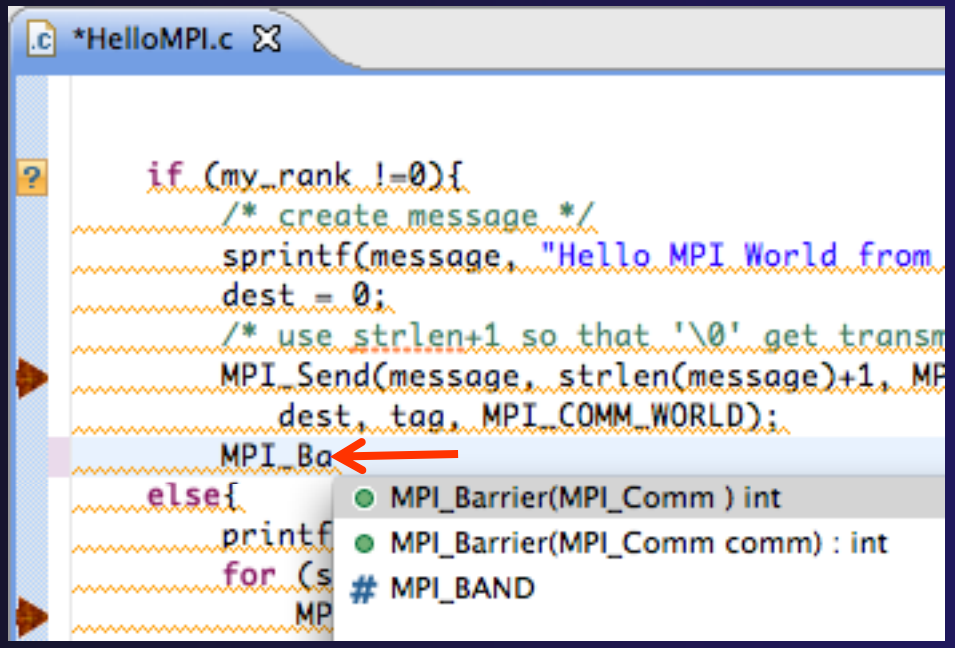
- For verified programs, lists barrier statements that synchronize together (match)
- For synchronization errors, reports counter example that illustrates and explains the error

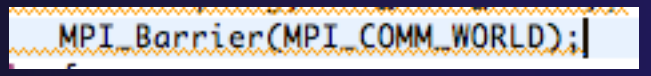


MPI Barrier Analysis – Try it

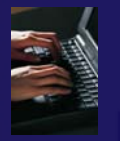
Add some barriers:

- Inside the sample if(rank...) add a barrier:
- Use Content Assist to help you type
- Type: MPI_ and press Ctrl-space. See completion alternatives. Keep typing until you see MPI_Barrier and hit enter.
- For args, start typing MPI_Comm_ etc. and it will also complete MPI_COMM_WORLD
- Add the same barrier statement at the end of the **else** as well.





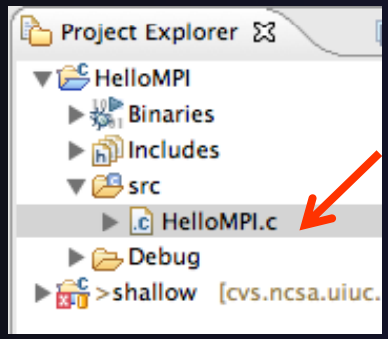
Resulting statement



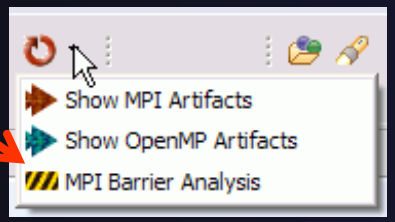
MPI Barrier Analysis – Try it (2)

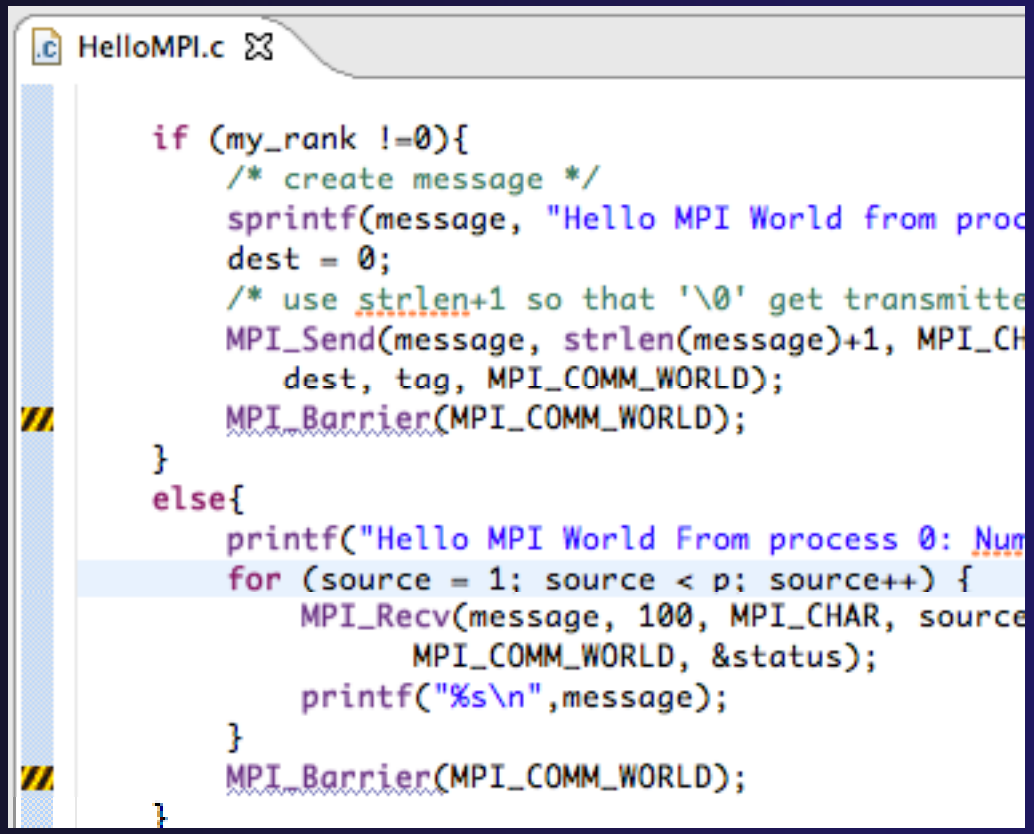
Run the Analysis:

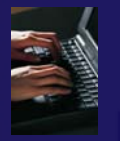
- In the Project Explorer, Select the source file (or directory, or project) of file(s) to analyze



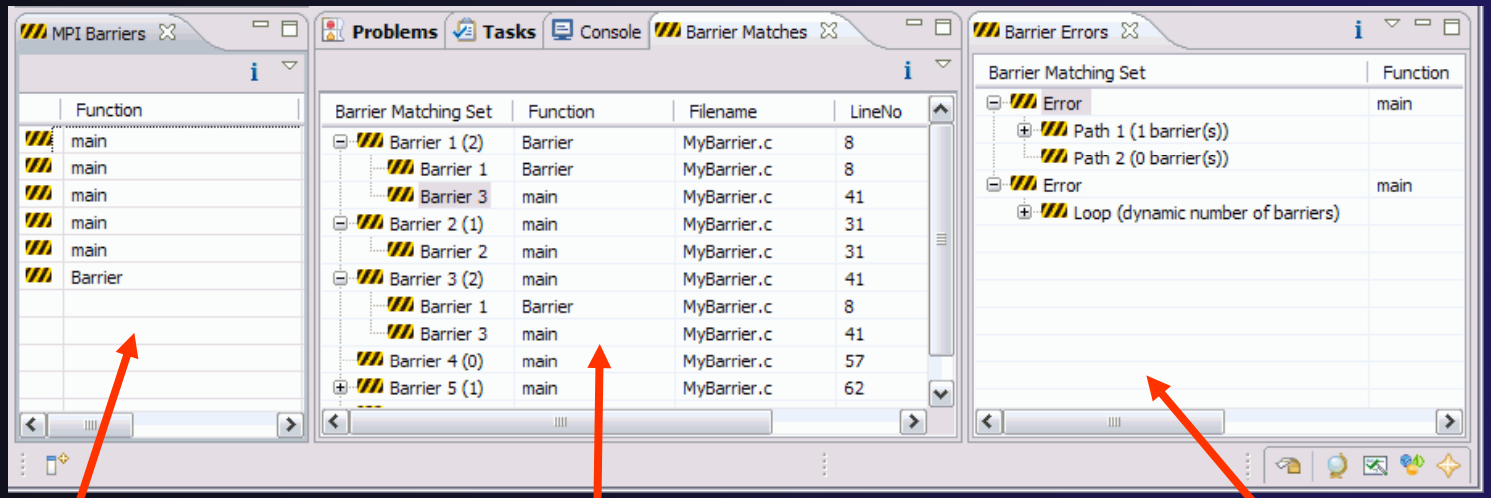
- Select the MPI Barrier Analysis action in the menu







MPI Barrier Analysis - views



MPI Barriers view

Simply lists the barriers

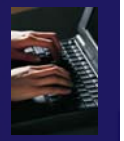
Like MPI Artifacts view, double-click to navigate to source code line (all 3 views)

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

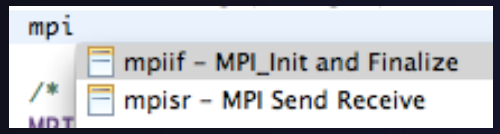
Barrier Errors view

If there are errors, a counter-example shows paths with mismatched number of barriers



MPI Templates

- Allows quick entry of common patterns in MPI programming
- Example: MPI send-receive
- Enter: mpisr <ctrl-space>
- Expands to the code shown at right
- Highlighted variable names can all be changed at once
- Type mpi <ctrl-space> <ctrl-space> to see all templates



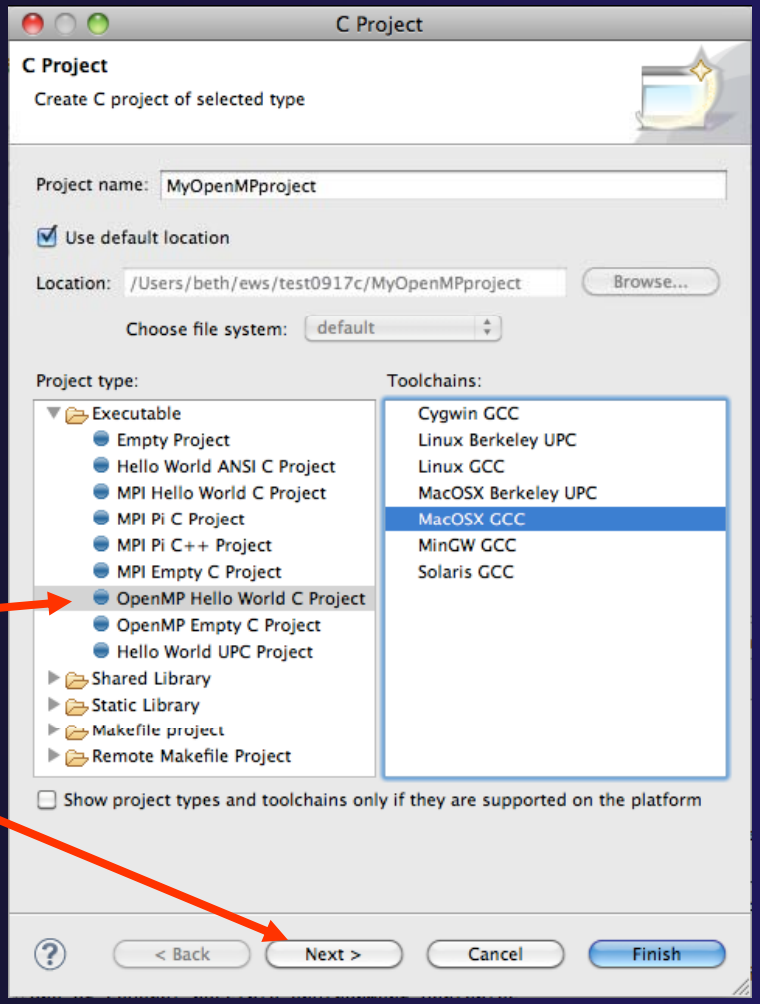

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);
if (rank == 0){ //master task
        printf("Hello  From process 0: Num processes: %d\n",p);
        for (source = 1; source < p; source++) {
            MPI_Recv(message, 100, MPI_CHAR, source, tag,
                  MPI_COMM_WORLD, &status);
            printf("%s\n",message);
        }
    }
    else{  // worker tasks
        /* create message */
            sprintf(message, "Hello  from process %d!", my_rank);
            dest = 0;
            /* use strlen+1 so that '\0' get transmitted */
            MPI_Send(message, strlen(message)+1, MPI_CHAR,
               dest, tag, MPI_COMM_WORLD);
    }
```

- Eclipse preferences: add more!
  - C/C++ > Editor > Templates
- Extend to other common patterns

OpenMP Managed Build Project

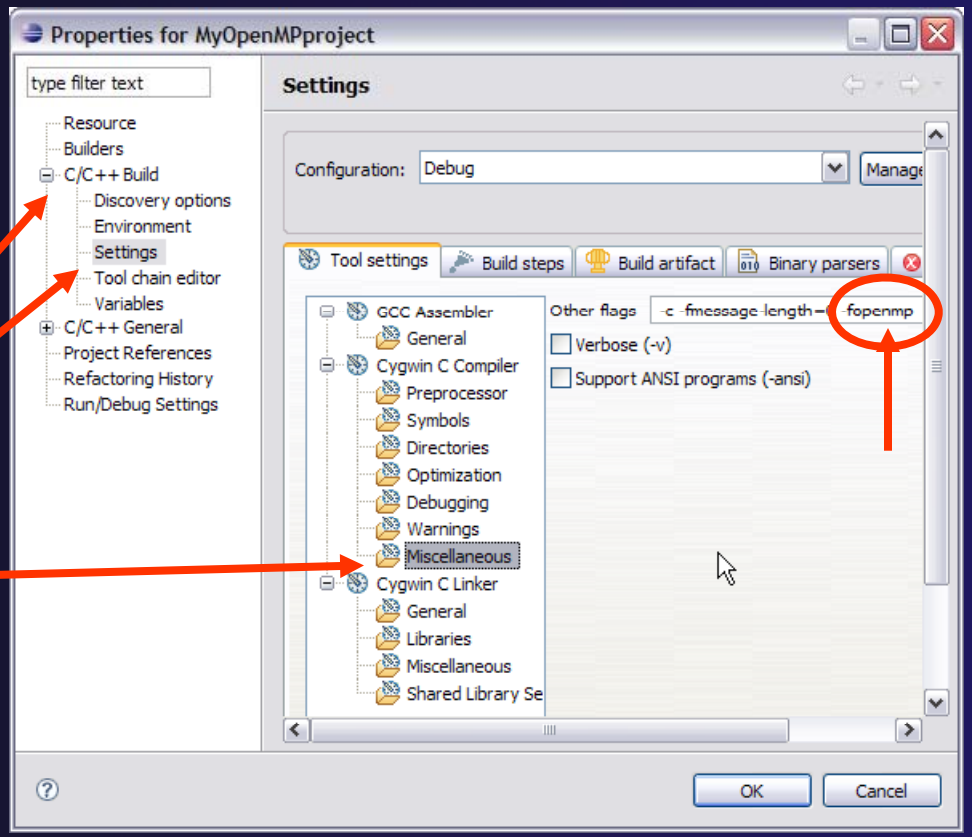
Local files only

- This will need OpenMP preferences (e.g. include file location) set up as well
- Create a new OpenMP project
  - **File ▸ New ▸ C Project**
  - Name the project e.g. 'MyOpenMPproject'
  - Select Toolchain
  - Select **OpenMP Hello World C Project**
  - Select **Next,** then fill in other info like MPI project



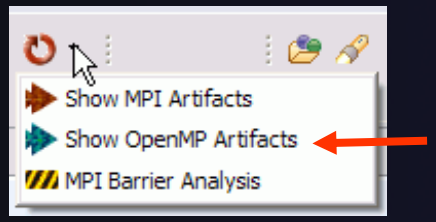
Setting OpenMP Special Build Options

- OpenMP typically requires special compiler options.
  - Open the project properties
  - Expand **C/C++ Build**
  - Select **Settings**
  - Select **C Compiler**
    - In Miscellaneous, add option(s). -fopenmp
- Click **OK**; Project should attempt to build

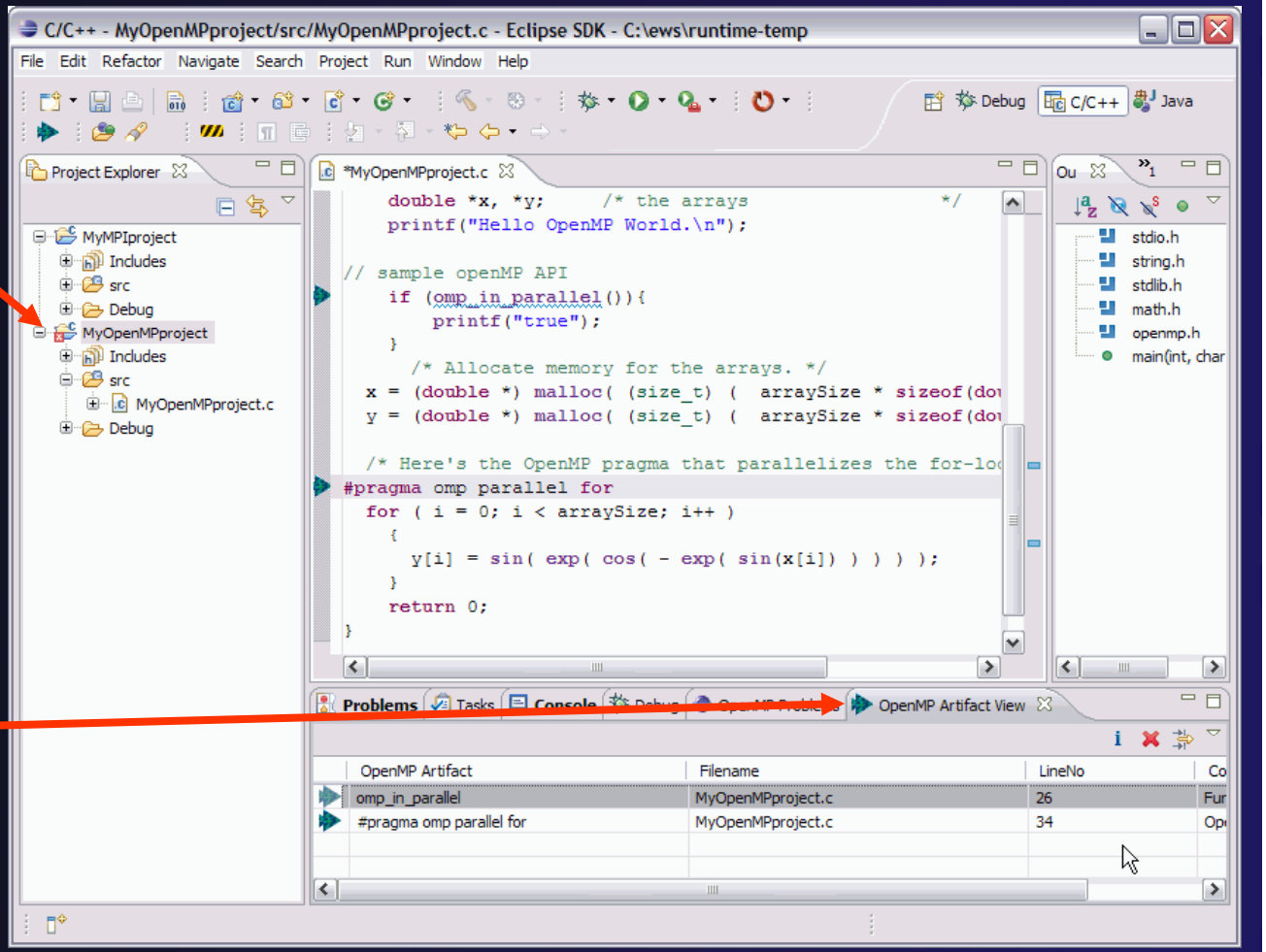


Show OpenMP Artifacts

- Select source file, folder, or project
- Run analysis

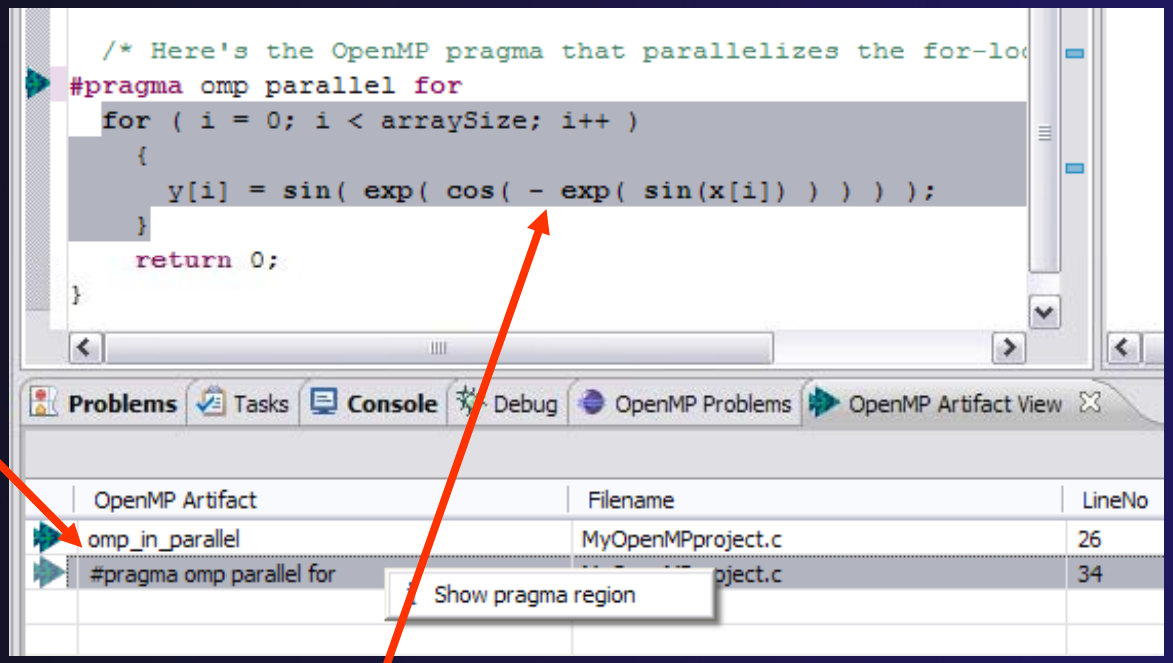


- See artifacts in **OpenMP Artifact view**



Show Pragma Region

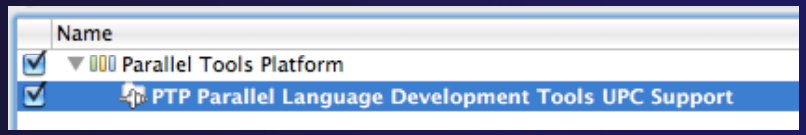
- Run OpenMP analysis
- Right click on pragma in artifact view
- Select **Show pragma region**
- See highlighted region in C editor



UPC

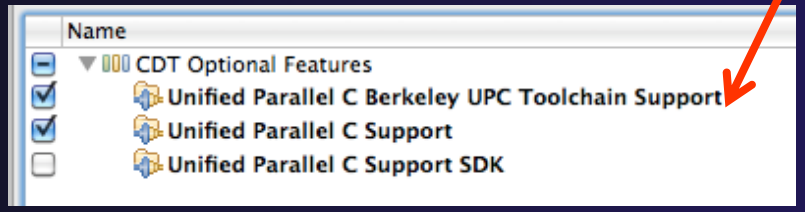
UPC Features Installation

- If you installed PTP PLDT UPC feature, you *should* have CDT UPC feature too



- See Also: http://wiki.eclipse.org/PTP/other_tools_setup#Using_UPC_features
- You can also install UPC features from the CDT-specific update site
  - Enable it in update manager
  - Help, Install New Software, Click **available Software Sites** link
  - Check the CDT site: http://download.eclipse.org/tools/cdt/releases/helios
  - Click OK to return to Install dialog
  - In **Work with:** select the CDT site you enabled
  - Check UPC features
  - Finish install and restart

BUPC toolchain only on CDT site



UPC syntax in .c files

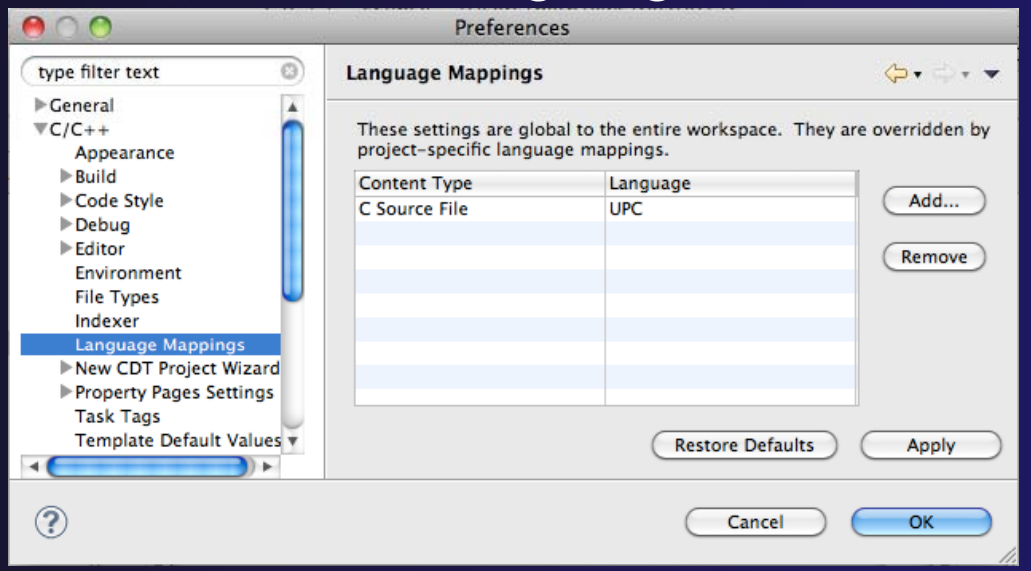
- UPC syntax is recognized by the parser in *.upc files
- Copy helloUPC.upc to hello.c to see the difference


```
// initialize the matrix a[][]
upc_forall (i=0; i<N; i++; &a[i][0])
  for (j=0; j<P; j++)
    a[i][j]=i*P+j+1;
```

Keywords as well as new syntax are recognized

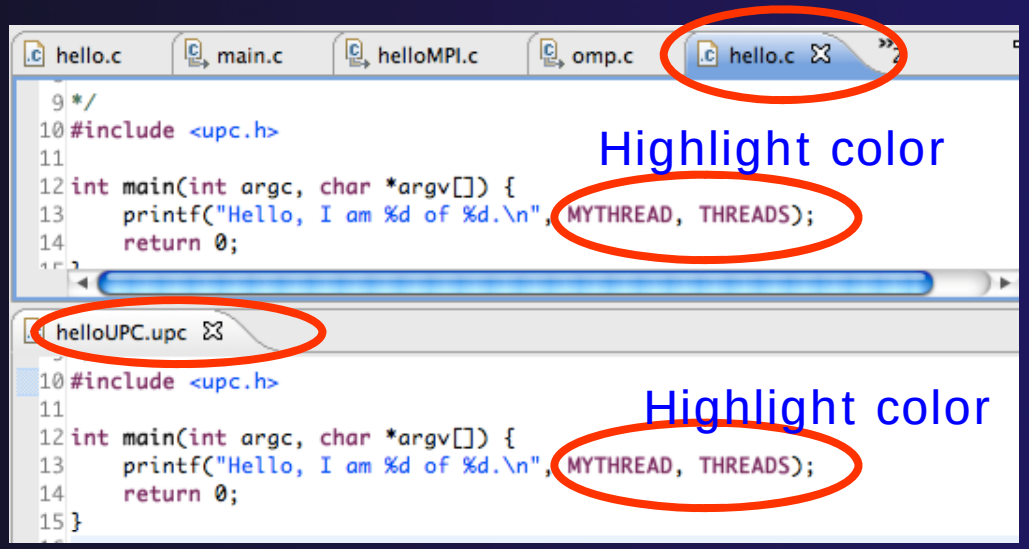
UPC syntax in .c files (2)

- To enable UPC syntax in *.c files, we will change the language mappings
- Preferences, C/C++, Language Mappings
- Click the **Add...** button to add a Language mapping.
- For Content Type, **C Source File**
- For Language, select **UPC**
- Click **OK**, **OK**



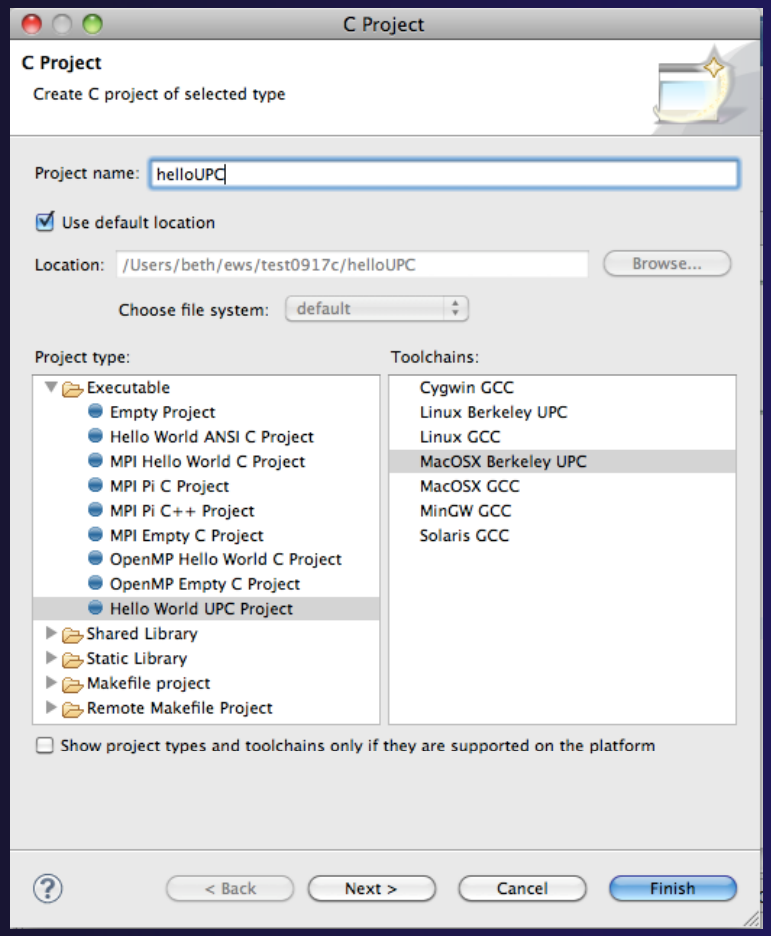
UPC syntax in .c files (3)

- Now UPC syntax is recognized in both types of files
- You may need to close and re-open a file to see the change.
- Note: in Project Properties, you can do this for just individual projects.



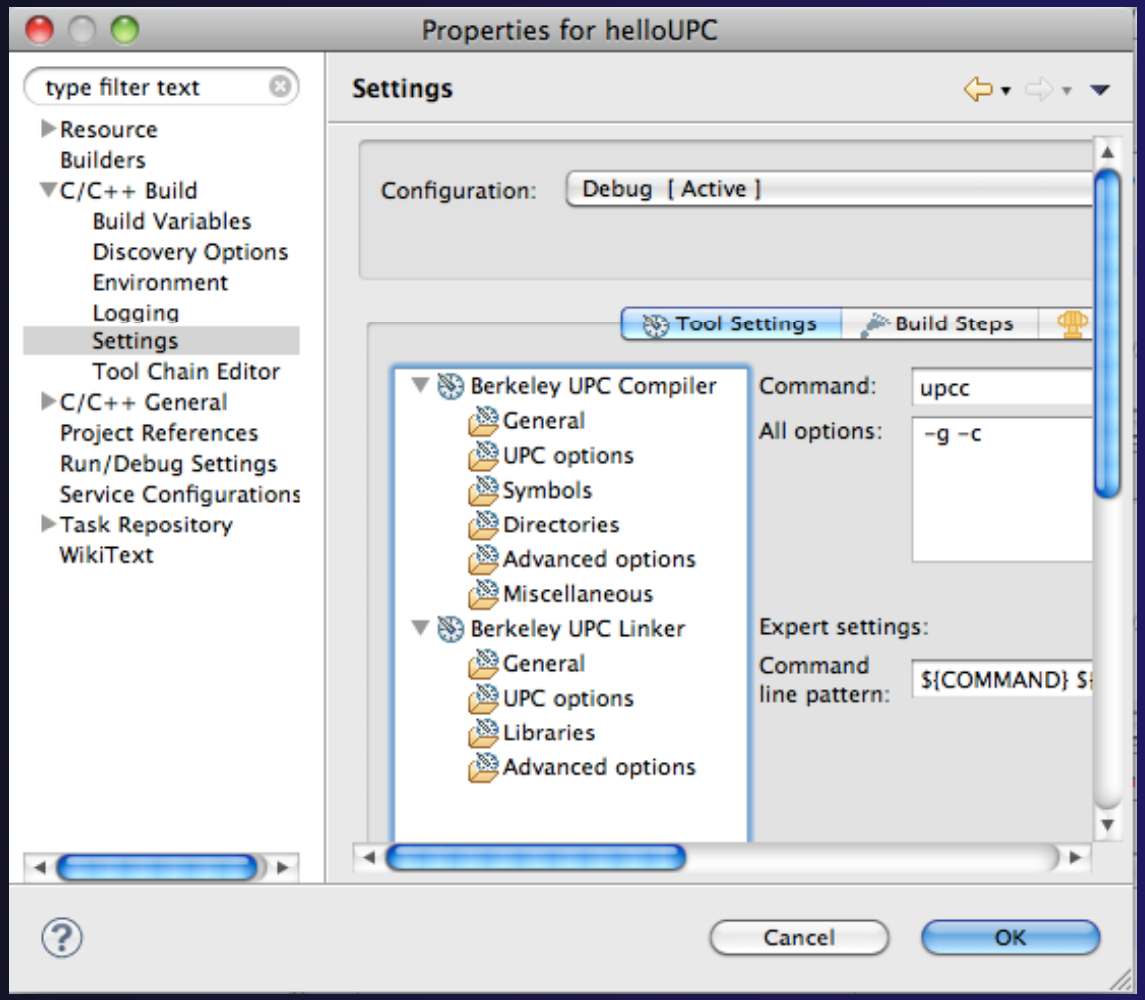
Berkeley UPC toolchain

- Local projects only
- File > New > C project
- Hello World UPC project
- Select toolchain (if you don't have the toolchain, it just won't build.)
- Next, Next, Finish



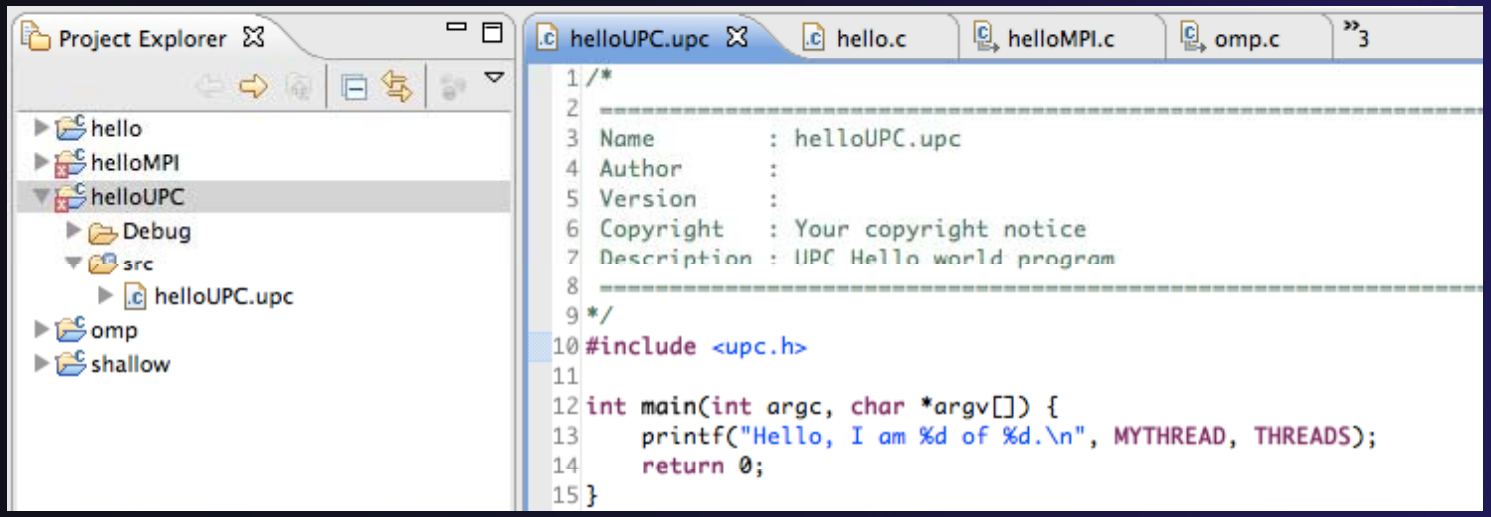
BUPC toolchain

- Bring up Project Properties to see details of BUPC toolchain:
- Project, right mouse, Properties



Hello World UPC project

- Hello (Berkeley) World UPC project
- Note UPC syntax highlighting
- Toolchain has been modified for UPC



UPC on abe.ncsa.uiuc.edu

- BUPC is located at:
  - /usr/apps/mpi/upc/berkeley_upc
- To run from cmd line on abe:
  - setenv PATH /usr/apps/mpi/upc/berkeley_upc/bin:${PATH}

TO RUN FROM PTP/ECLIPSE:

- In your home dir on abe: use ‘helloUPC’ to make a remote proj
- Set Remote Paths and Symbols to include:
  - /usr/apps/mpi/upc/berkeley_upc/opt/include/upcr_preinclude
- To run: use a Generic Remote Launch for Resource Manager
- Run config:
  - Application program: /usr/apps/mpi/upc/berkeley_upc/bin/upcrun
  - Arguments tab: -q -n 4 ~/helloUPC/helloUPC

External Tools Framework ETFw Motivation

- There are numerous command-line oriented development tools employed in HPC
- These can be complicated or time consuming to use
- IDE integration for individual development tools is slow and inconsistent
- We want all our development tools in one place with one interface
- We want our development tools to work together

ETFw: Development Tool Workflows

- Variations on ‘Compile, Execute, Analyze-Results’ are common to most software development
- These steps may be tedious and time consuming, especially over multiple iterations
- By defining both tool interfaces and behavior in an XML document these steps can be simplified and automated

ETFw: The Build Phase

```
    <compile>
<!--  By default the compiler commands set here prepend whatever compiler is already in use in Eclipse.  If you set the tag
replace="true" for the compile element the compilers will be replaced entirely with the command specified here. Each compiler type,
c, c++ and fortran, is defined as shown below.  -->
<!-- Every command referencing a file on the system should include a group tag.  The group tag indicates that the relevant binary files
or scripts are located in the same place for each command sharing that tag  -->
        <CC command="vtcc" group="vampirtrace">
<!-- Arguments to be passed to a command may be specified with the argument tag as shown here. -->
            <argument value="-vt:cc"/>
        </CC>
        <CXX command="vtcxx" group="vampirtrace">
            <argument value="-vt:cxx"/>
        </CXX>
        <F90 command="vtf90" group="vampirtrace">
            <argument value="-vt:f90"/>
        </F90>
    </compile>
```

- Set compilers and arguments for each language
- Define UI for compiler/compiler-wrapper configuration

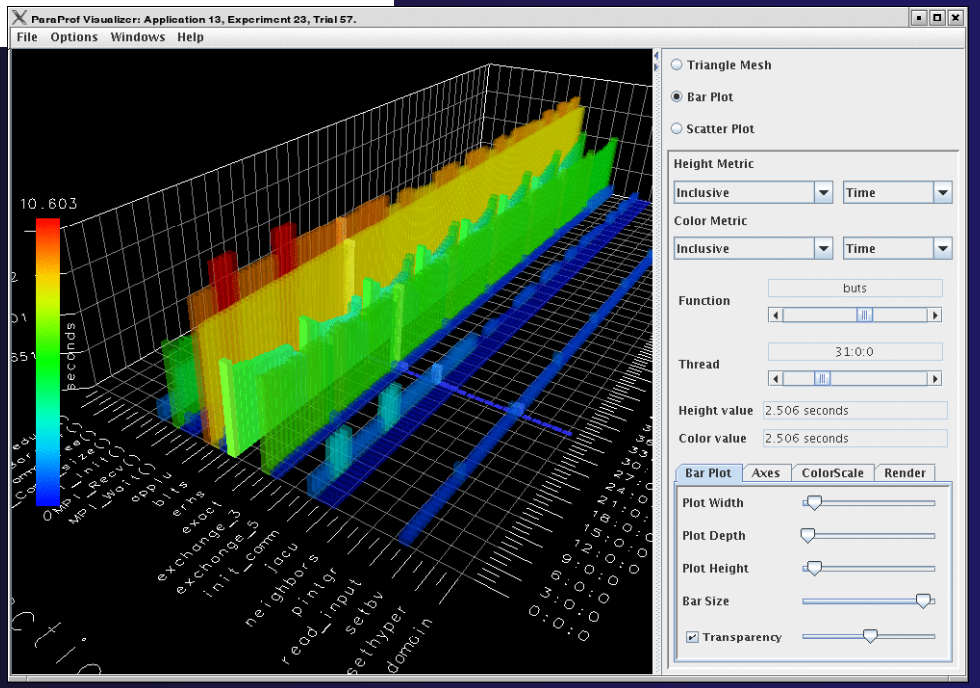
ETFw: The Execution Phase

```
<execute>
    <utility command="mpirun" group="mpi">
        <argument value="-np 4"/>
    </utility>
    <utility command="psrun" group="perfsuite">
    </utility>
</execute>
```

- Specify composed execution tools such as Perfsuite or Valgrind
- Set launch environment variables
- Define variables and tool options in XML or provide a UI in the IDE
- Integrates with PTP parallel launch environment

ETFw: The Analysis/Post-Processing Phase

```
<analyze>
    <utility command="expert" group="kojak">
        <argument value="a.elg"/>
    </utility>
    <utility command="paraprof" group="tau">
        <argument value="a.cube"/>
    </utility>
</analyze>
```

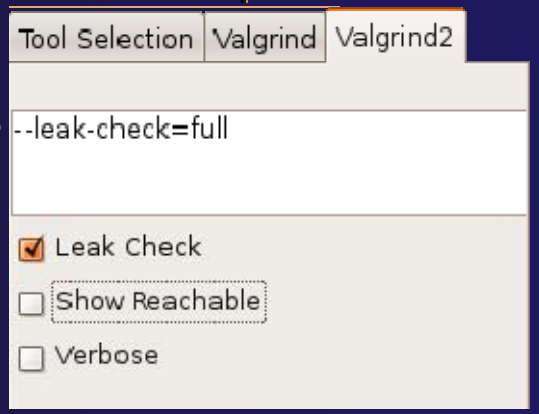


- Sequentially run tools on program output
- Launch external visualization tools

ETFw: XML-Defined UI Components

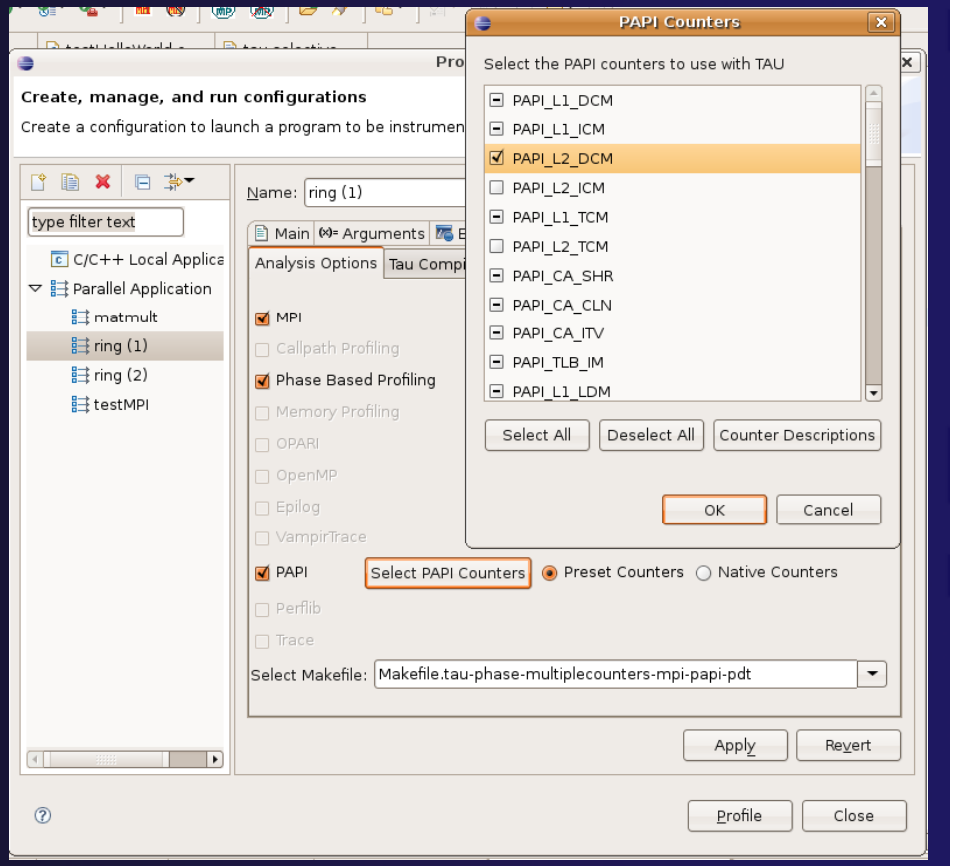
```
<tool name="Valgrind2">
    <execute>
        <utility command="bash" group="inbin"/>
        <utility command="valgrind" group="valgrind">
            <optionpane title="Valgrind2" seperatewith=" ">
                <togoption label="Leak Check" optname="--leak-check=full" tooltip="Full memory leak check" defstate="true"/>
                <togoption label="Show Reachable" optname="--show-reachable=yes" tooltip="Show reachable units"/>
                <togoption label="Verbose" optname="--verbose" tooltip="Verbose output"/>
            </optionpane>
        </utility>
    </execute>
</tool>
```

- Each pane constructs a set of options sent to a tool or a set of environment variables
- Numerous options for converting a command line interface into an intelligent GUI without Eclipse coding



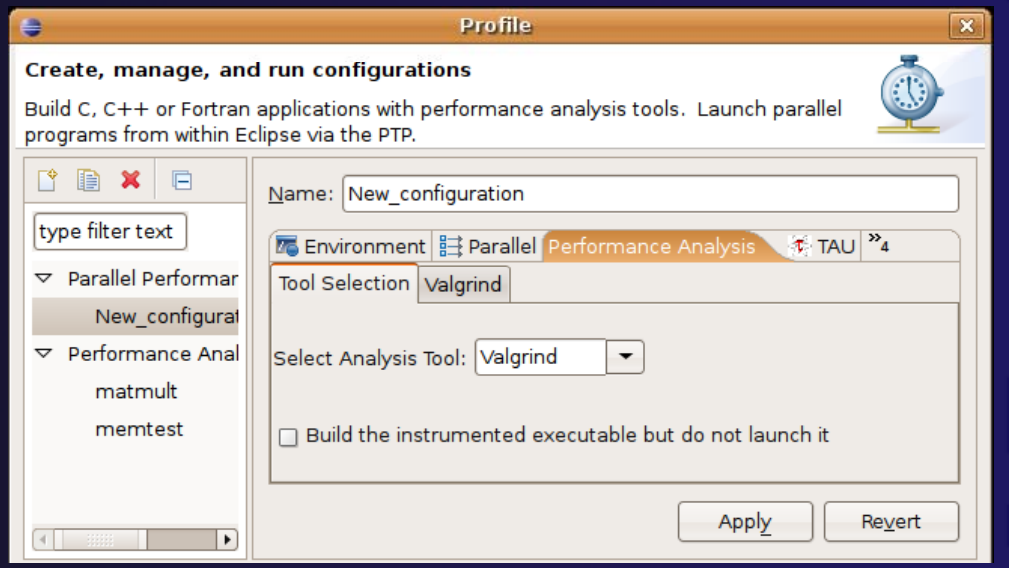
ETFw: Advanced Components

- Extension points allow integration with UIs and workflow behavior too complex to define in XML
- Logical and iterative workflows for successive executions and parametric studies

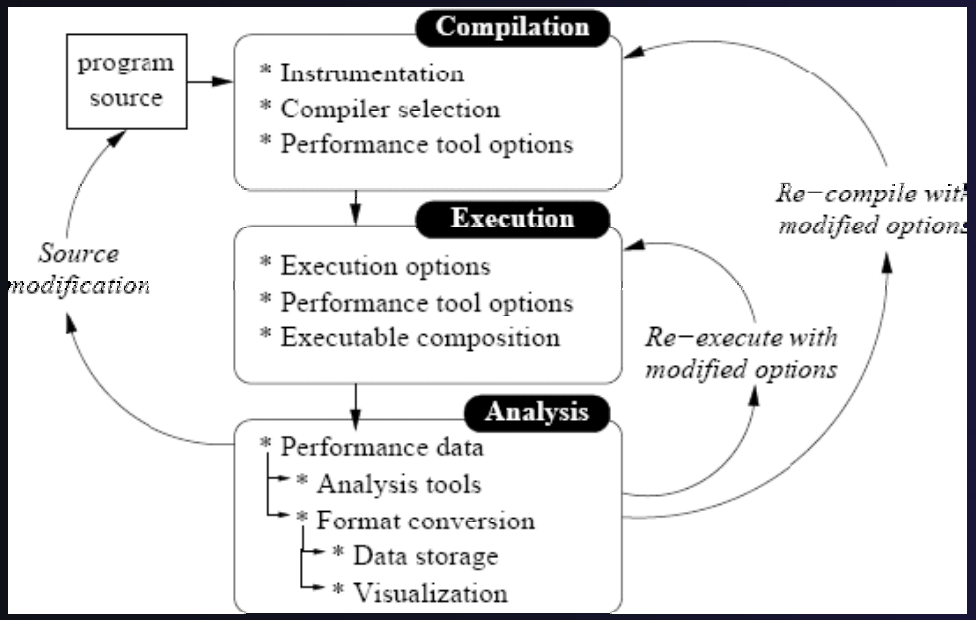


ETFw: Using Workflows

- New workflows are added to the ETFw launch configuration system
- Multiple workflow configurations can be defined and saved for different use cases
- XML Workflow definitions can be saved and reused in different environments



ETFw: General Purpose Workflow



- Automated
- Generalized
- Quick performance analysis and other development tool integration
- Exposes tool capabilities to the user

ETFw: Continuing Development

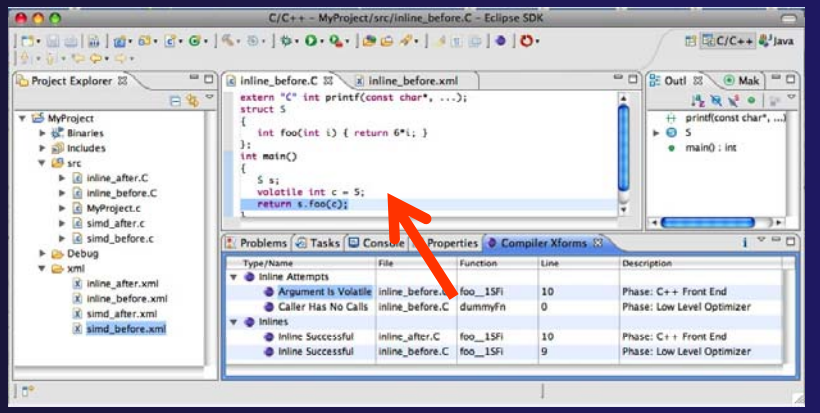
Plans:

- Integration with PTP Remote Development Tools
- Additional options for GUI definition
- Generalization of TAU specific features such as hardware counter selection and performance data storage

- Contact: Wyatt Spear

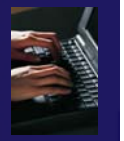
ETFw Feedback view

- Many existing tools provide information that can be mapped to source code lines
  - Compiler errors, warnings, suggestions
  - Performance tool findings
- ETFw feedback view provided to aid construction of these views
  - Currently geared toward data provided by tools in XML files
- Original ETFw facilities aid the CALL of external tools from PTP
  - Feedback view aids the exposition of results to the user



Examples:

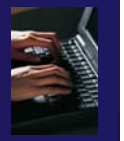
- Compiler optimization report
- Performance tool data
- Refactoring tool uses “advice” from external files



Feedback Sample

- Download a sample implementation of the feedback view:
- Complete instructions here: http://wiki.eclipse.org/PTP/ETFw/feedback

- And on following slide...



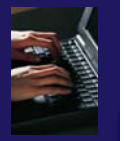
Feedback Sample – (1) Install

- Download the plugin jar file
- http://download.eclipse.org/tools/ptp/misc/feedback/org.eclipse.ptp.etfw.feedback.sample_1.0.0.201010280927.jar

- Save it in your eclipse/dropins directory
  - This is a “quick and dirty” type of installation
  - Eclipse knows to look here when it starts, and it installs whatever it finds here
- Then restart eclipse
  - You should see the feedback icon

Feedback Sample – (2) data files

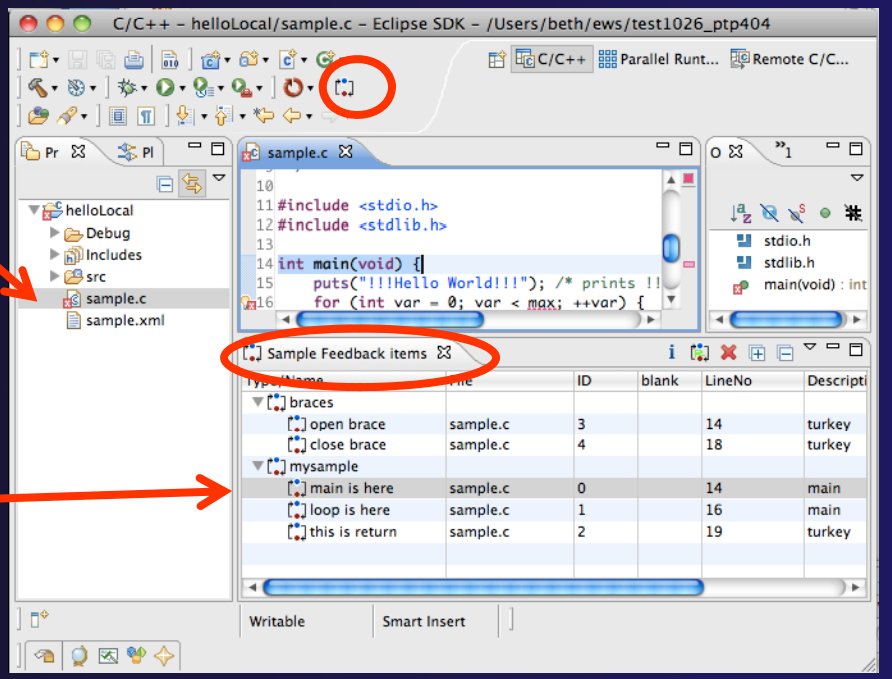
- You have the Feedback sample plug-in installed
- Now you need some sample files for it to process
  - sample.c and sample.xml
  - They are hidden in the plug-in!
  - Let's take it apart to find them
  - Unzip the jar file; they are in the data/ directory
    - Alternate instructions on the wiki page
  - Put them in a (local) eclipse project



Feedback Sample – (3) Try it

- You have the Feedback sample plug-in installed
- You have an xml file that it can parse, and the source file that it refers to.

1. Select xml file
2. Click feedback button
3. See Sample Feedback view
4. Double-click in view to navigate to source code lines



END