Application Development with Eclipse and the Parallel Tools Platform

Greg Watson & Craig Rasmussen
Los Alamos National Laboratory

PART 2

LAUR-05-7669

Parallel Tools Platform

Contents:

- Overview
- Installing PTP
- Building Parallel Programs
- Launching Parallel Programs
- Monitoring Parallel Resources
- Parallel Debugging
- MPI Development Tools

Parallel Tools Platform

OVERVIEW

Overview

- Project Objectives
  - Extend Eclipse to support parallel application development
    - Not MPI specific
  - Equip Eclipse with key tools needed to start developing parallel codes
    - Parallel runtime, parallel debugger, Fortran, etc.
  - Encourage parallel tool developers to support Eclipse
    - MPI Development Tools
  - Develop a new generation of parallel tools needed to meet the demands of HPCS

Overview

- Project History
  - Prototype demonstrated in Nov 2004
  - Project formally approved by Eclipse Foundation in Feb 2005
  - Project launched at EclipseCon in March 2005
  - First milestone met in June 2005
  - IBM joins PTP effort in September 2005
  - Version 1.0 due end October 2005

Overview

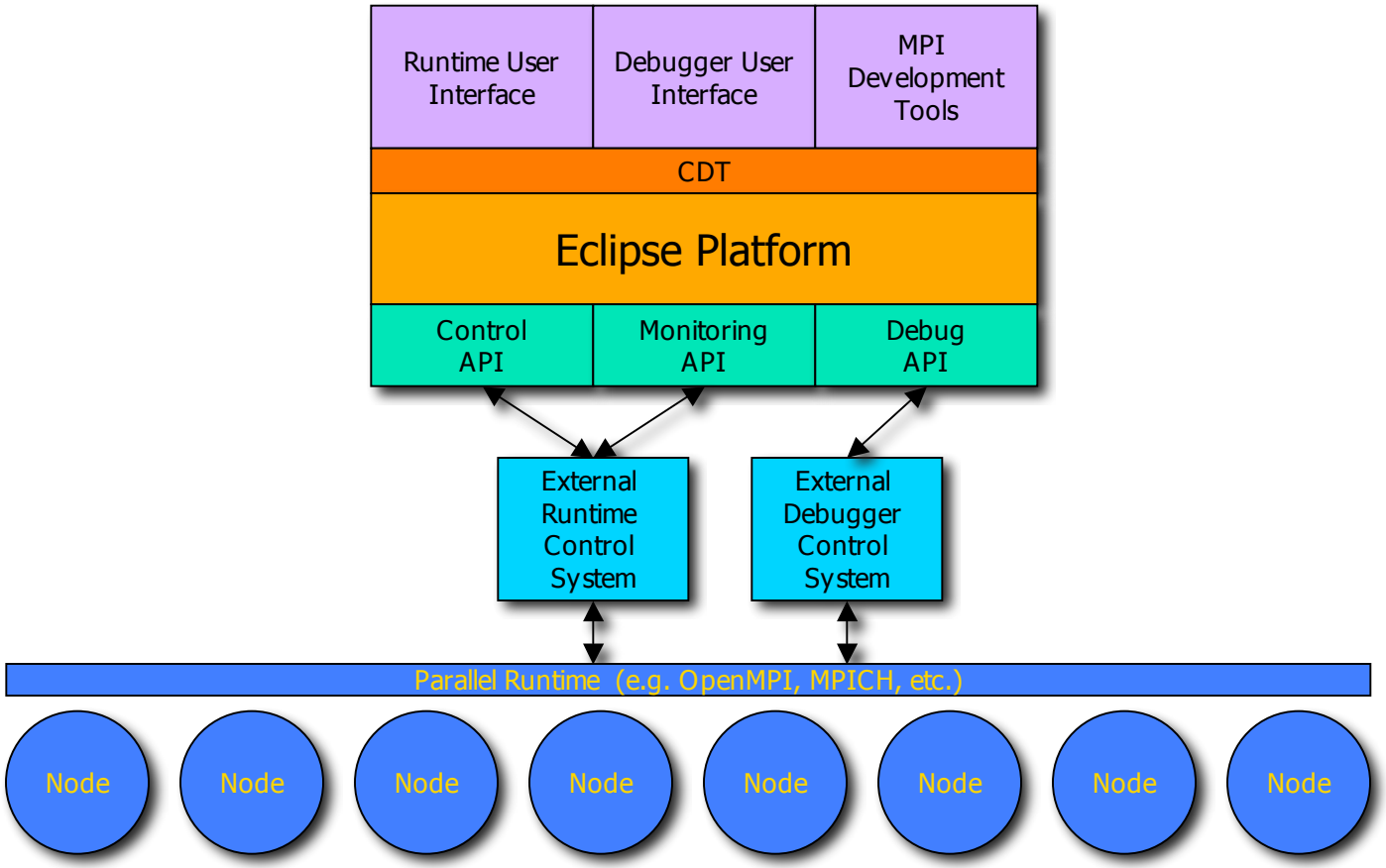
- Project Collaborations
  - Languages
    - University of Illinois
  - MPI Tools
    - IBM Research
    - Argonne National Laboratory
    - Open MPI
  - Performance Tools
    - University of Oregon
    - Technischen Universität München
    - Livermore National Laboratory
    - Sandia National Laboratory

Overview

- PTP Components
  - Parallel runtime
    - User interface
    - External API
    - External monitor/controller
  - Parallel debugger
    - User interface
    - External API
    - External debug manager
  - Infrastructure services for parallel tools
  - MPI development tools

Overview

- PTP Architecture



INSTALLING PTP

Installing PTP

- Download from PTP web site
  - http://eclipse.org/ptp/downloads
- PTP Core
  - org.eclipse.ptp-1.0.0-RC1.tar.gz
- PTP External Components (available soon)
  - org.eclipse.ptp-ompi-1.0.0-RC1.tar.gz
  - org.eclipse.ptp-sdm-1.0.0-RC1.tar.gz
- MPI Development Tools
  - org.eclipse.ptp-mpi-1.0.0-RC1.tar.gz

Installing PTP

- Core and MPI Tools
  - Extract files
  - Copy the contents of features to the features folder of your Eclipse installation
  - Copy the contents of plugins to the plugins folder of your Eclipse installation
  - Restart Eclipse
    - Use `-clean` option if new components don't appear
- External Components
  - Installation depends on architecture and runtime
  - Come to next workshop!

BUILDING PARALLEL PROGRAMS

Building Parallel Programs

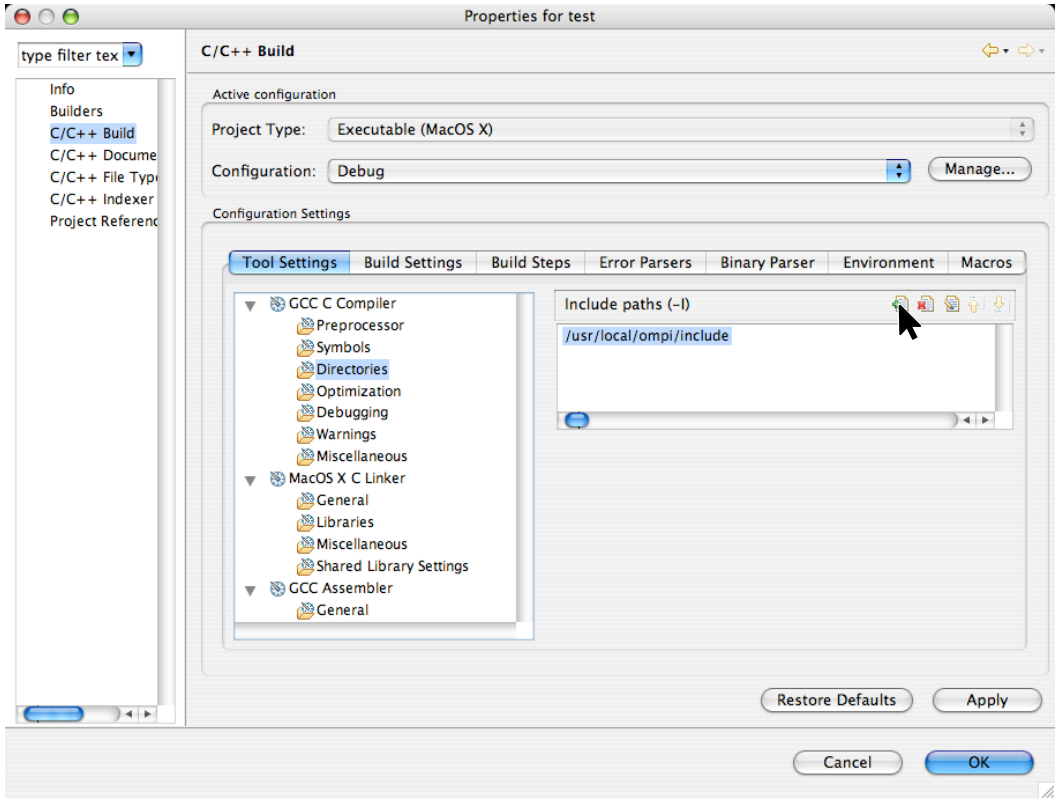
- Standard Make Project
  - Use an existing (or new) Makefile to build the project using the appropriate compiler (e.g. mpicc)
  - Has the advantage of preserving existing build process
  - Does not take advantage of the Eclipse build system

Building Parallel Programs

- Managed Make Project
  - Create a managed make project
  - Open project properties
  - Select C/C++ Build (or Make/Build)
  - Make sure the Tool Settings tab is selected

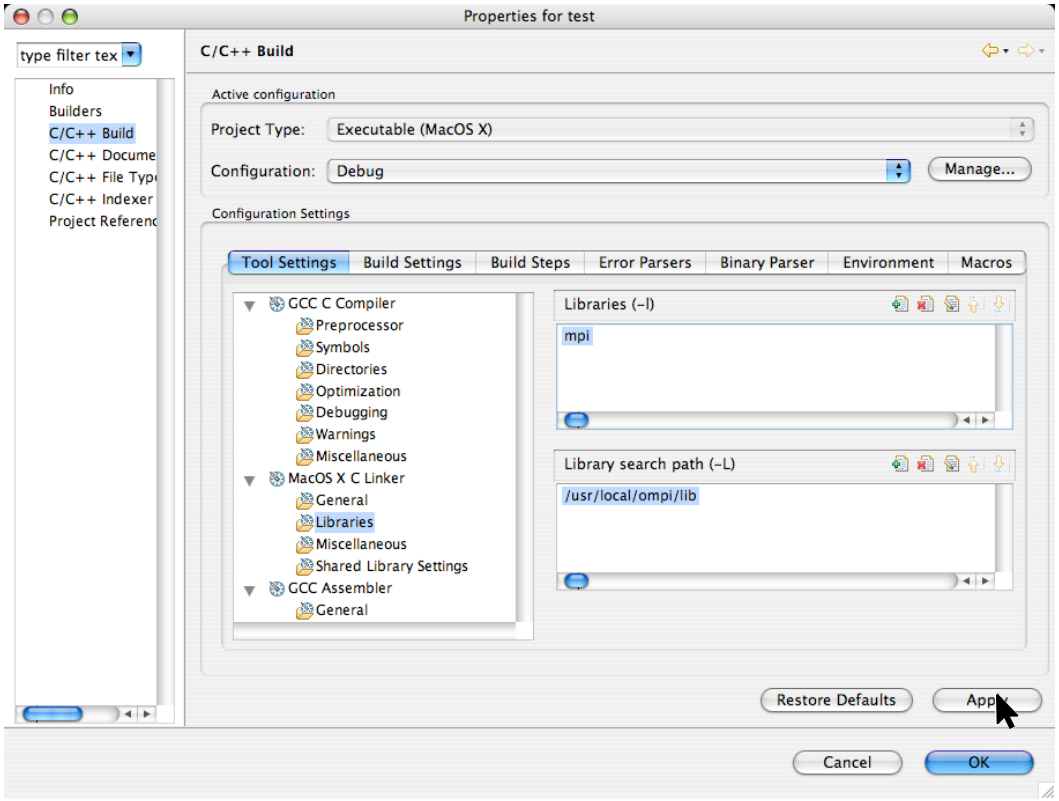
Building Parallel Programs

- Add MPI include path to C Compiler



Building Parallel Programs

- Add library and library path to C Linker



LAUNCHING PARALLEL PROGRAMS

Launching Parallel Programs

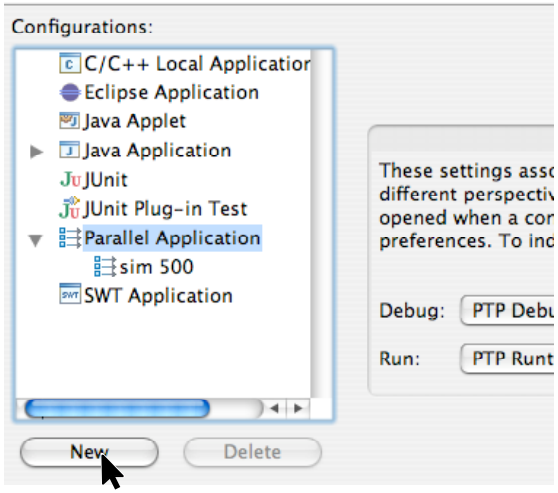
- The Old Way:
  - Open the run configuration dialog (Run…)
  - Create a new C/C++ Local Application (or Local Application)
  - Choose the project
  - Enter the *full path* of the `mpirun` command in the C/C++ Application (or Application) field
  - Select the Arguments tab
  - Enter the `mpirun` arguments and the name of the executable
  - Select Apply and then Run

Launching Parallel Programs

- Limitations of the Old Way:
  - No indication of job or process status
  - Cannot easily interact with a job scheduler
  - Must have MPI installed on local machine
  - No access to individual stdout of processes
  - No ability to debug program

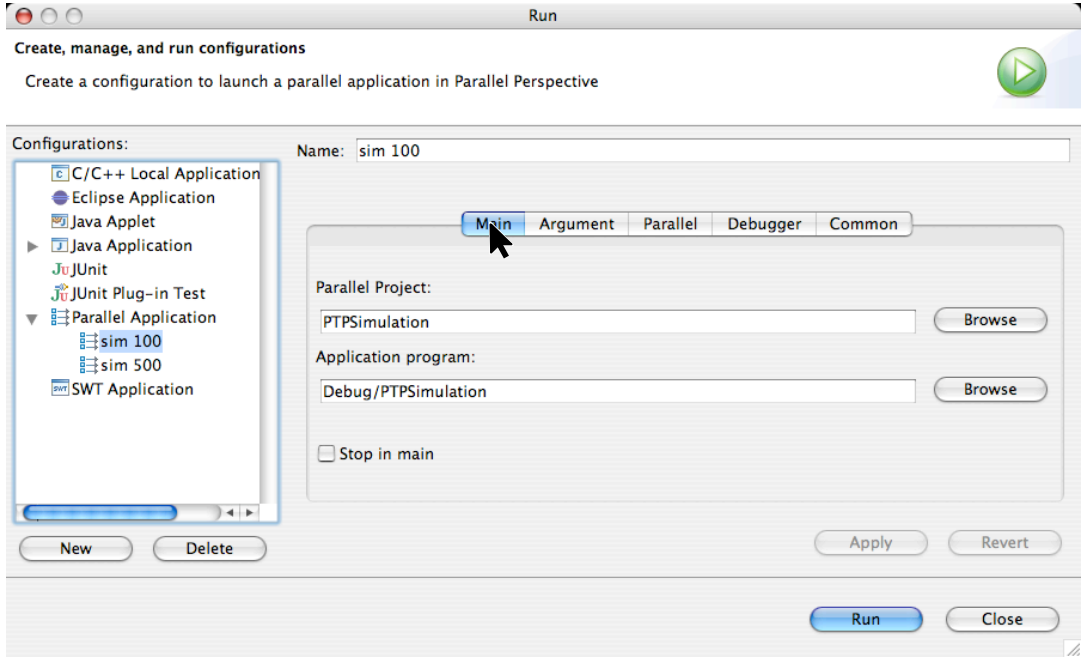
Launching Parallel Programs

- The New Way:
  - Open the run configuration dialog (Run…)
  - Create a new Parallel Application



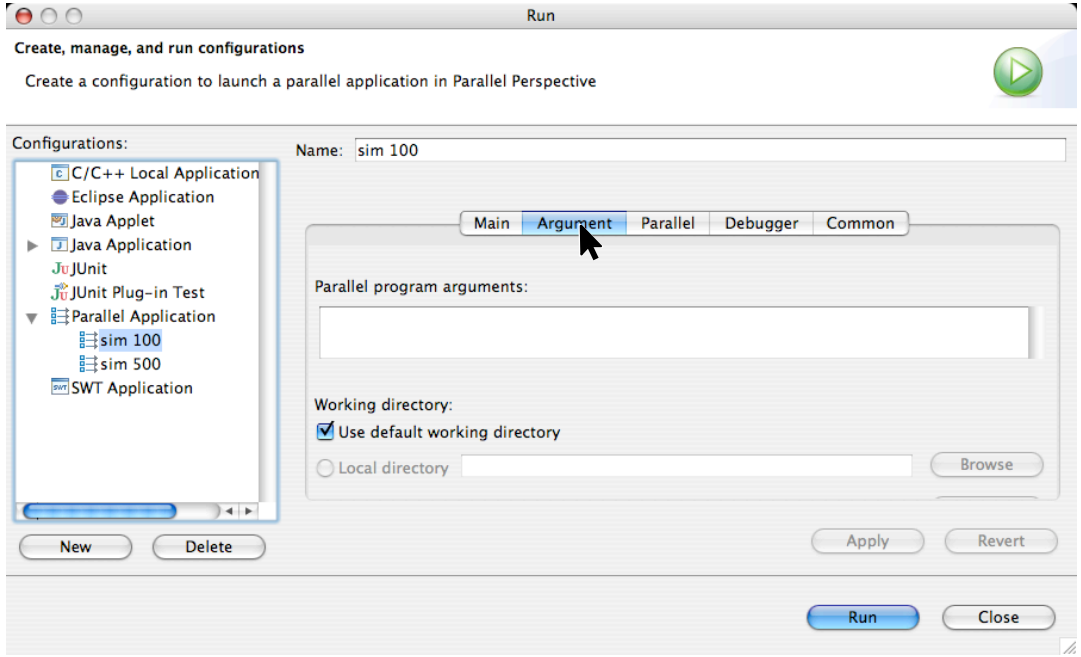
Launching Parallel Programs

- In the Main tab
  - Enter a name for the configuration
  - Choose the project
  - Choose the executable from the project



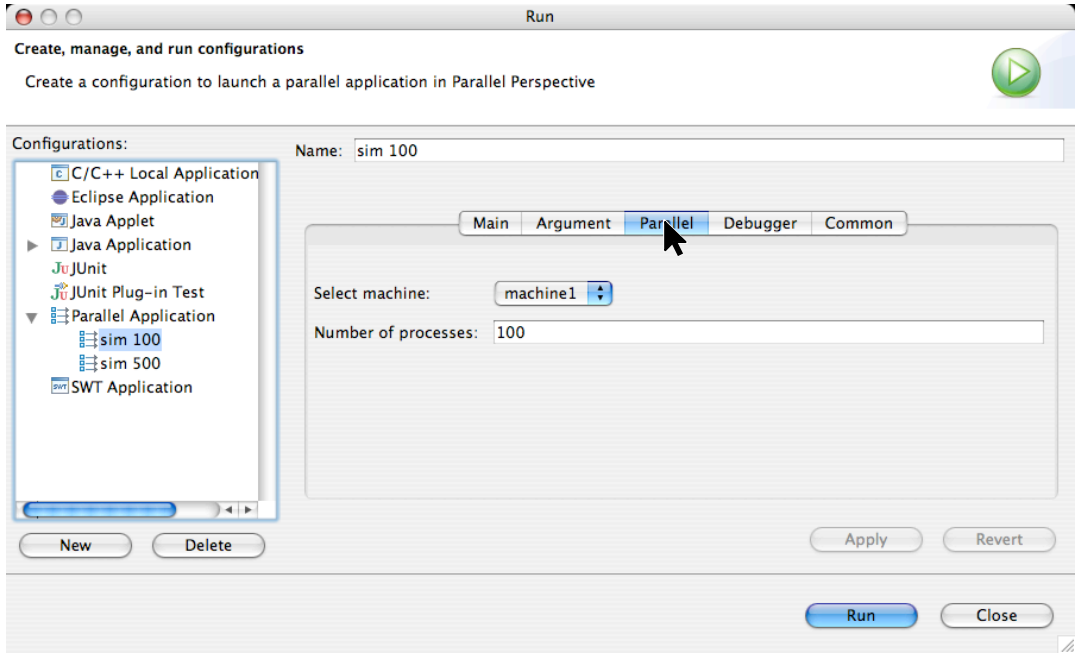
Launching Parallel Programs

- Select the Arguments tab
  - Enter any *program* arguments
  - Change working directory if necessary



Launching Parallel Programs

- Select the Parallel tab
  - Choose the machine that the job will run on
  - Enter the number of processes to run
  - Click on Apply, then Run



MONITORING PARALLEL RESOURCES

Monitoring Parallel Resources

- Open the PTP Runtime perspective
  - Window → Open Perspective → Other…
  - Select PTP Runtime
  - PTP will automatically query runtime for available resources

Monitoring Parallel Resources

- Main features of the PTP Runtime perspective

Resource Views

Resource Detail

Process Detail

Process Output

PTP Runtime – job0_process51 – Eclipse SDK

Debug | PTP Runtime | C/C++ | PTP Deb

Machines | Jobs

machine1 – Root [256]

job0_process51

Process details

Rank: 51 | PID: 54901000 | Node: 51

Job: 0 | Total: 100

Status: stopped | Exit code:

Program output

Console | Problems | Properties | Error Log

0 errors, 7 warnings, 0 infos

Description	Resource	In Folder	Lo...on
return type defaults to 'int'	test.c	array_test	line 6
unused variable 'a'	test.c	array_test	line 9
control reaches end of non-	test.c	array_test	line 1
C/C++ Indexer Problem: Pre	main.c	test	line 2
implicit declaration of funct	main.c	test	line 1
implicit declaration of funct	main.c	test	line 1

Node Info

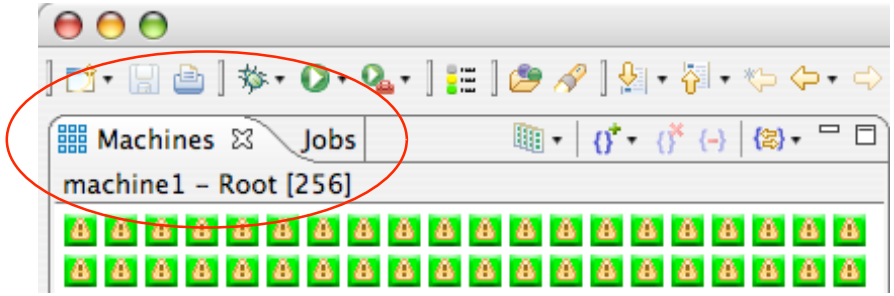
Node	51
State	up
User	greg
Group	ptp
Mode	0100

Process Info

Process 51, Job 0

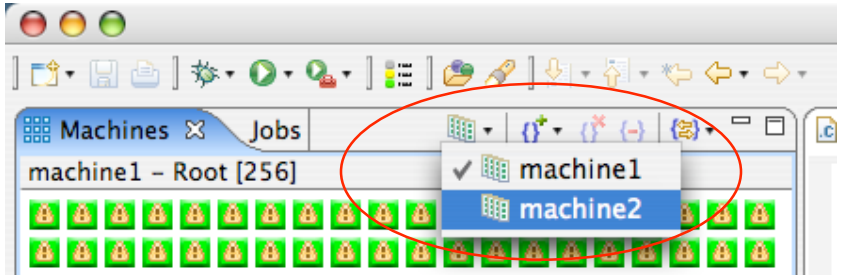
Monitoring Parallel Resources

- Choosing resources to view
  - Select between Machines view or Jobs view

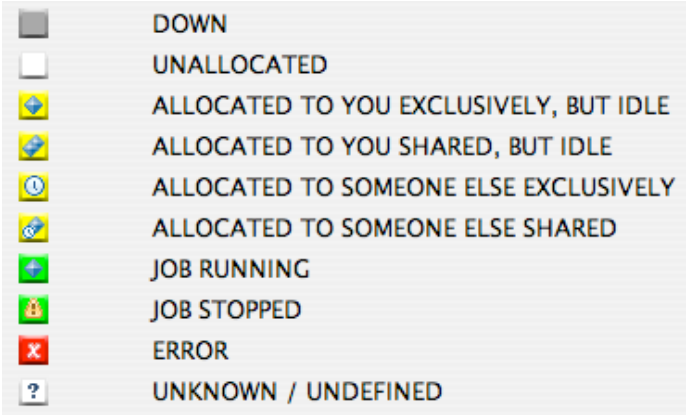


Monitoring Parallel Resources

- In Machines view
  - Switch between machines using the dropdown menu

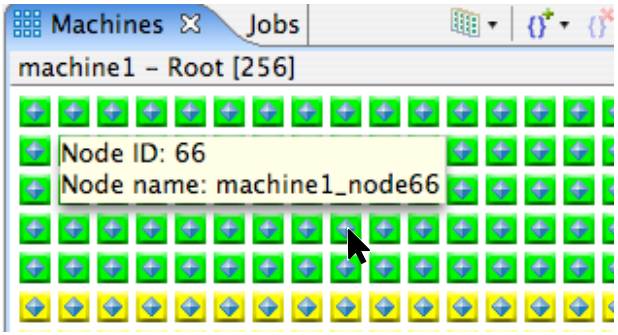


  - Node status indicated by icons

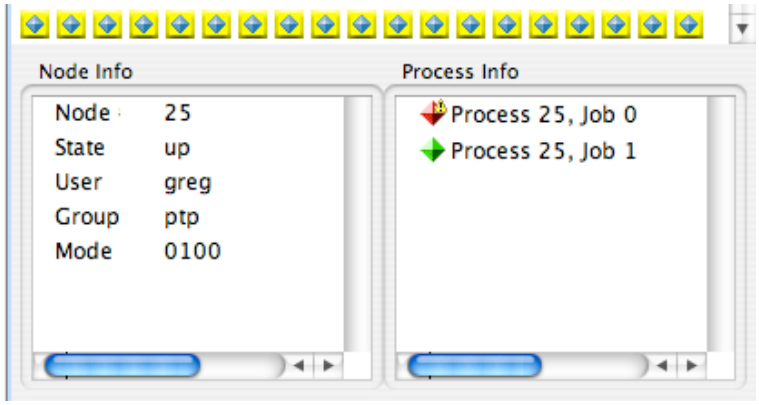


Monitoring Parallel Resources

- Hovering over node shows ID and name

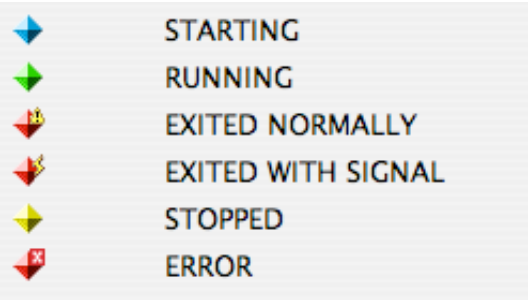


- Double click on node to show more information



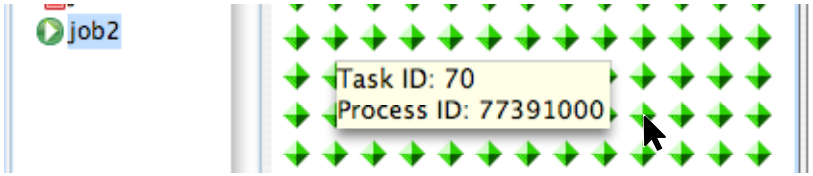
Monitoring Parallel Resources

- In Jobs view
  - Running and completed jobs are listed
  - Selecting a job shows processes belonging to job
  - Process status indicated by icon

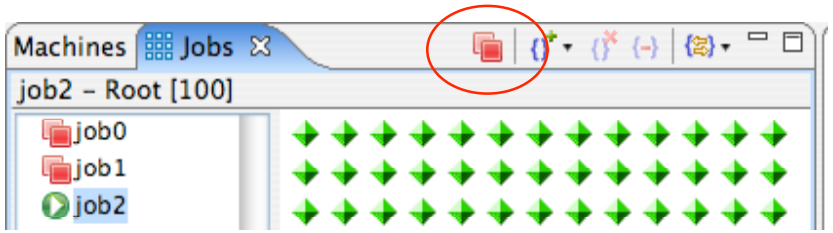


Monitoring Parallel Resources

- Hovering over process shows MPI task ID and process ID



- Terminate button can be used to kill job



Parallel Tools Platform

- Future Plans (Building, Launching and Monitoring)
  - Ability to specify additional job resource requirements
  - "Generic" job scheduler interface
  - Support for delayed job launching
  - Disconnect/reconnect to running jobs
  - Remote build support
  - Remote job launch, monitoring and control
  - Data management

PARALLEL DEBUGGING

Parallel Debugging

- Main features of the PTP Debug perspective

Jobs View

Current Process Set

Process View

Source View

Breakpoints View

PTP Debug – main.c – Eclipse SDK

Debug | PTP Runtime | C/C++ | PTP Debug | Java

Parallel Debug

job3 – Root [100]

job3

Breakpoints | Expressions | Variables | Array View

Root

main.c [line: 12] {Job: Global – Root}

main.c [line: 15] {Job: Job: job3 – Root}

Debug

Process 0 (Suspended)

Process 51 (Suspended)

Thread [51 – 0] (Suspended)

1 main() at main.c:13

Debugger

Launch Label 0

main.c

```
#include <stdio.h>

void display() {
    const int s1 = 10;
    const int s2 = 20;

    int array1[s1];
    int array2[s1][s2];
    int num = 10;
    int i,j = 0;

    for (i=0; i<s1; i++) {
        array1[i] = i;
        for (j=0; j<s2; j++) {
            array2[i][j] = i + j;
            printf("Array: %d\n", array2[i][j]);
        }
    }
```

Outline

stdio.h

display

main

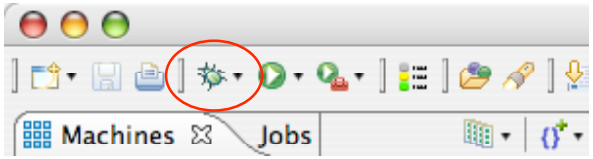
Error Log | Console

simulation [Parallel Application] Launch Label 51

Los Alamos NATIONAL LABORATORY EST. 1943

Parallel Debugging

- Launching a debug session
  - Create a Parallel Application in the usual way
  - Select Stop in main to automatically stop the job
  - Select Debugger tab to change any default settings (not normally required)
  - Select Apply, then Debug
  - For existing Parallel Application launch configurations, launch using the debug button rather than the run button



  - Eclipse will switch to the PTP Debug perspective

Parallel Debugging

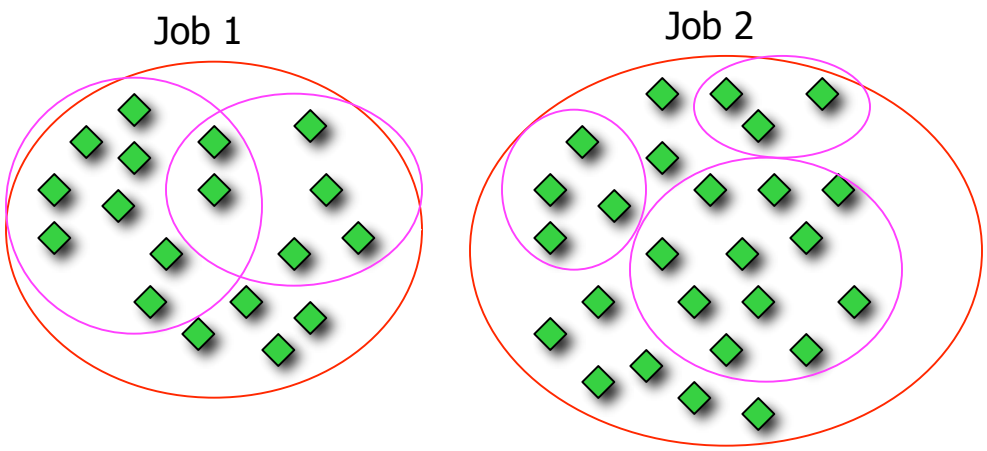
- Process sets
  - Traditional debuggers apply operations to a single processes
  - Parallel debugging operations apply to single process *or to arbitrary collections of processes*

Definition:

- A *process set* is a means of simultaneously referring to one or more processes

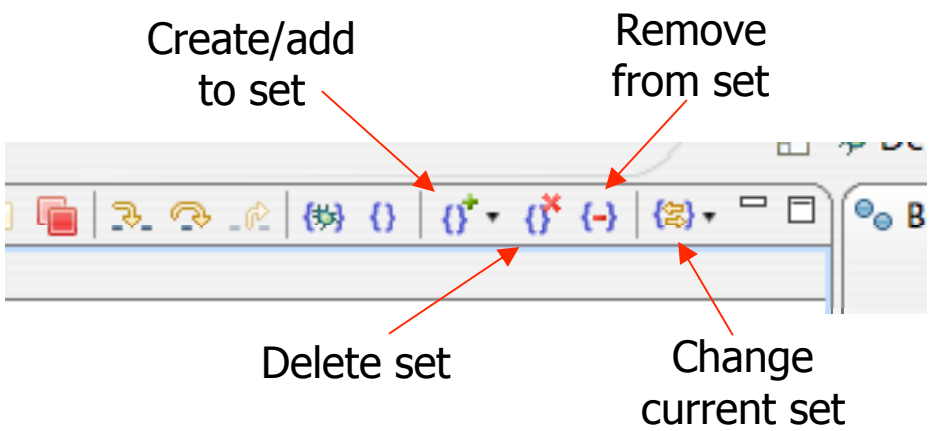
Parallel Debugging

- Process Sets (cont…)
  - 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

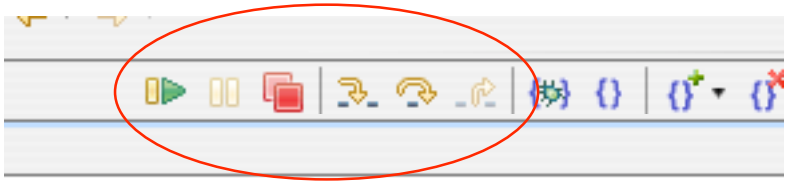


Parallel Debugging

- Process set operations

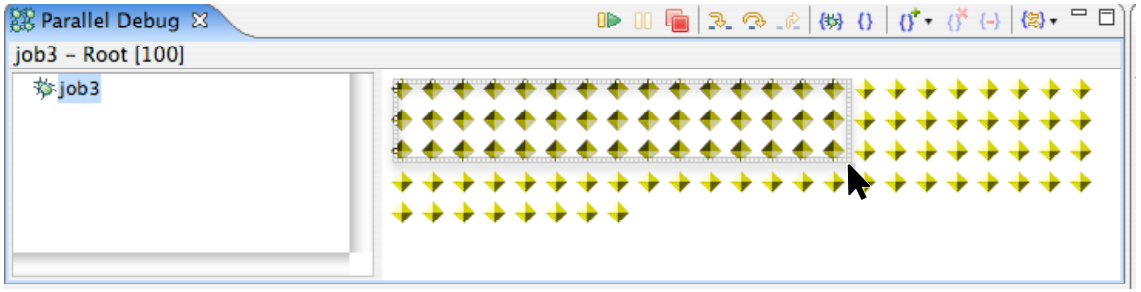


- Debug operations always apply to the current set

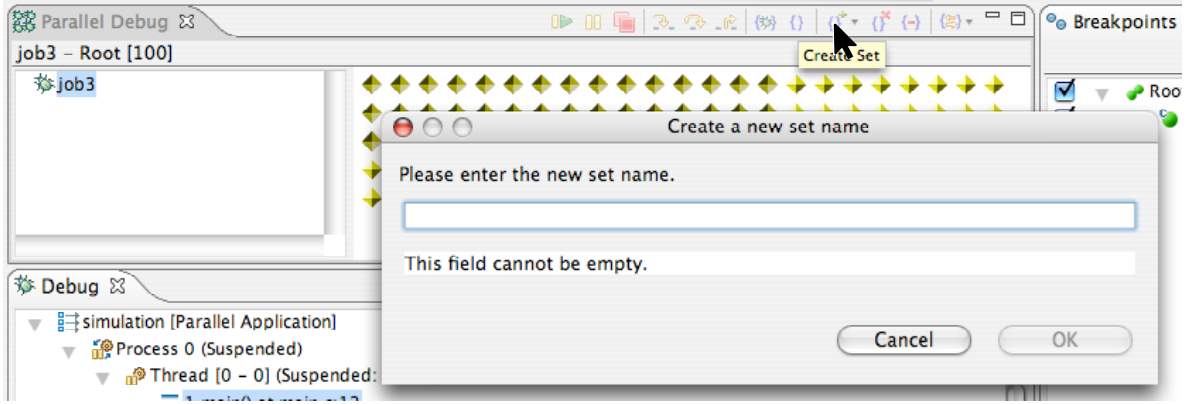


Parallel Debugging

- Creating process sets
  - Select the processes to be placed in the set

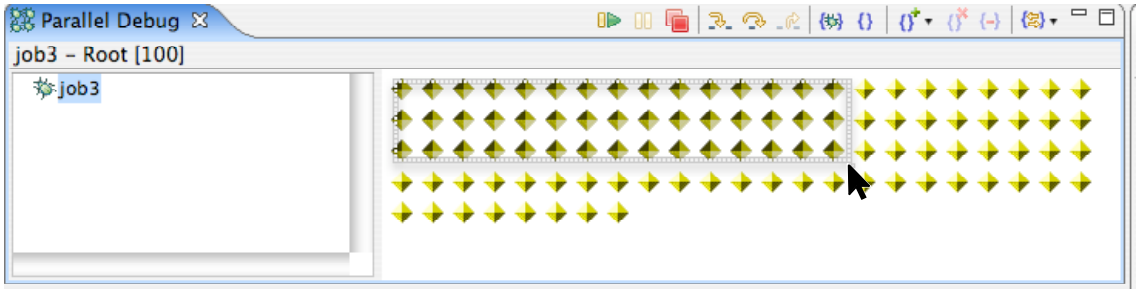


  - Select create process set button and enter a name

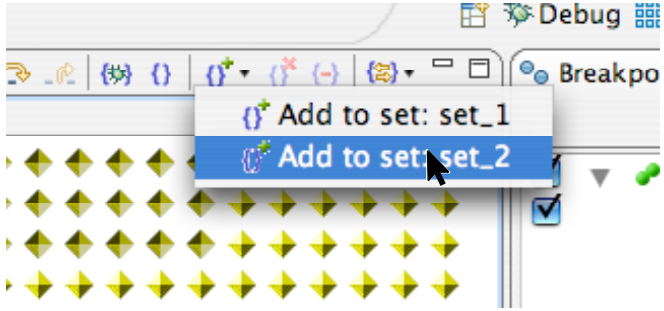


Parallel Debugging

- Adding processes to a set
  - Select the processes to be added to the set

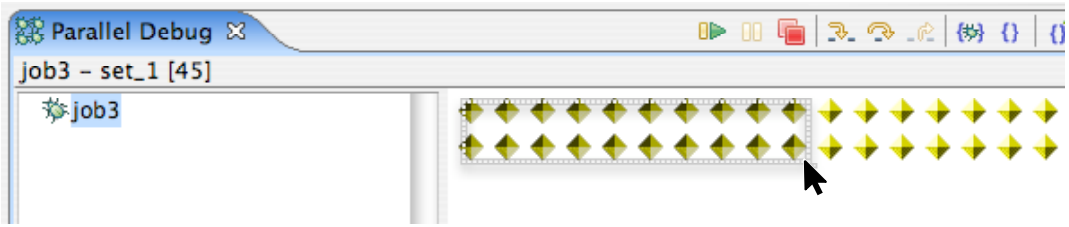


  - Choose set to add processes to

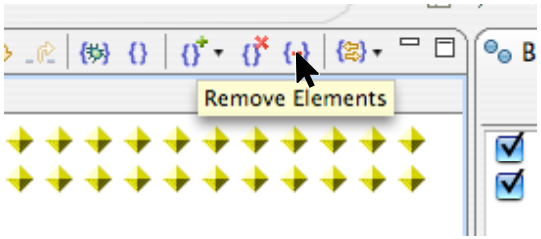


Parallel Debugging

- Removing processes from a set
  - Select the processes to be removed from the set

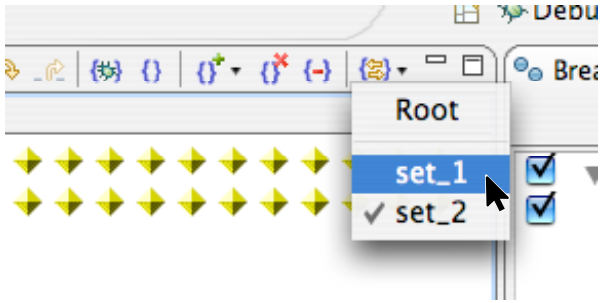


  - Remove the processes

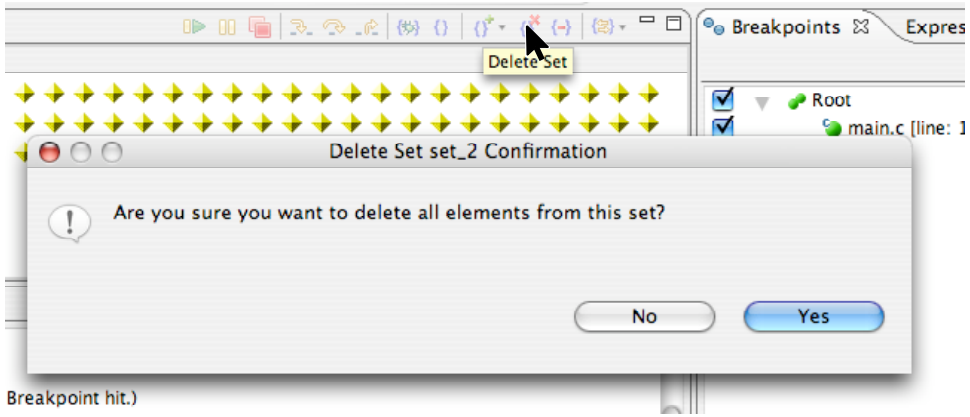


Parallel Debugging

- Changing current process set (clicking on button will cycle through sets)

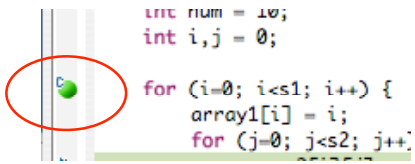


- Deleting current process set

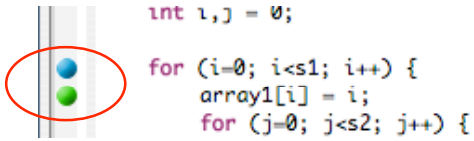


Parallel Debugging

- Breakpoints
  - There are two main types of breakpoints
  - *Global breakpoints*
    - Apply to *all* processes in *any* job

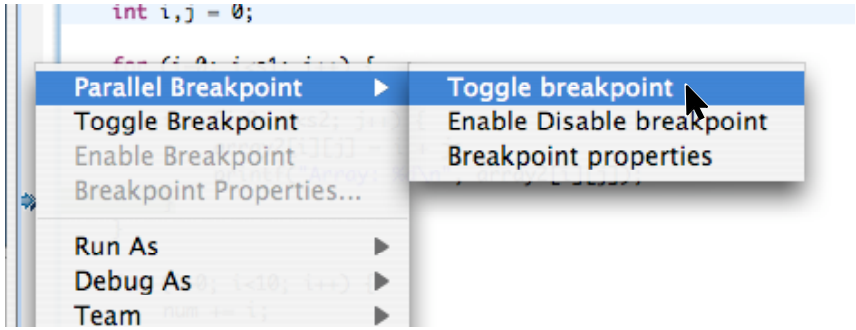


  - *Set breakpoints*
    - Apply only to process in a particular set for a single job
    - Green indicates breakpoint applies to current set, blue to some other set



Parallel Debugging

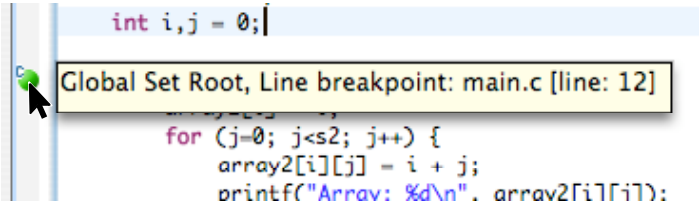
- Creating breakpoints
  - Double click on left hand edge of an editor window
  - Right click and use context menu



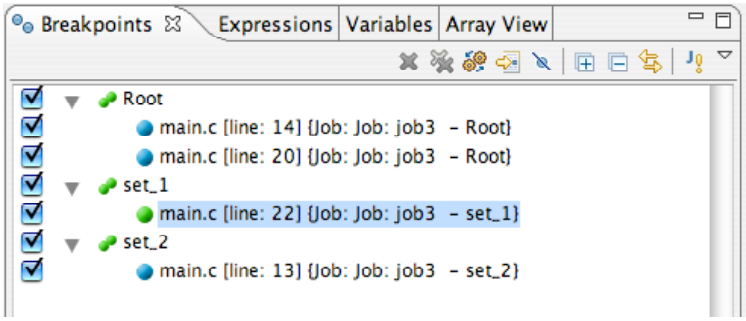
  - A global breakpoint is created if no jobs are selected
  - If a job is selected the breakpoint will apply to the current set

Parallel Debugging

- Breakpoint information
  - Hover over breakpoint to see more information

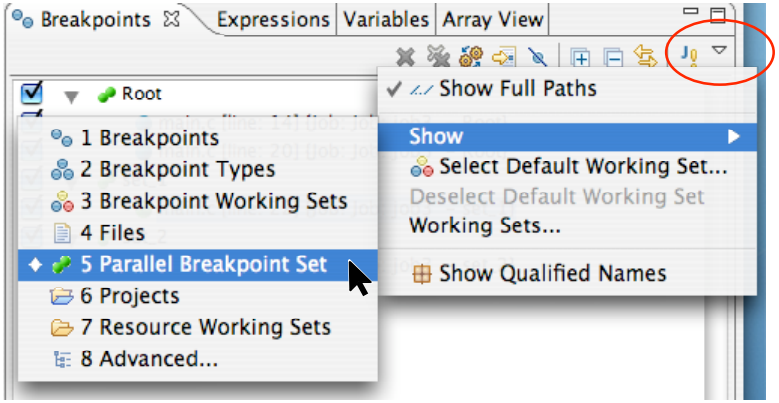


  - Use Breakpoints tab to see all breakpoints



Parallel Debugging

- Use menu to group breakpoints by type



Parallel Debugging

- Current Instruction Pointer
  - Used to show current location of *suspended* process
  - Traditional programs
    - single instruction pointer (the exception to this is multi-threaded programs)
  - Parallel programs
    - an instruction pointer for every process
  - PTP debugger
    - one instruction pointer for *every group of processes at the same location*

The group of processes represented by an instruction pointer in not necessarily the same as a process set

Parallel Debugging

- Single instruction pointer in normal debugger

```
 * @param args
 */
public static void main(String[] args) {
    System.out.println("about to start...");
    test xxx = new test();
    xxx.run();
}
```

Current source line is also highlighted

- Multiple instruction pointers in PTP debugger

Un-registered process marker

Multiple markers

Registered process marker

```
int i,j = 0;

for (i=0; i<s1; i++) {
    array1[i] = i;
    for (j=0; j<s2; j++) {
        array2[i][j] = i + j;
        printf("Array: %d\n", array2[i][j]);
    }
}
```

Parallel Debugging

- Hovering over instruction pointer provides additional information

```
int num = 10;
int i,j = 0;

for (i=0; i<s1; i++) {
    array1[i] = i;
}

for (i=0; i<10; i++) {
    num += i;
```

Multiple markers at this line
-Suspended on registered process: 0
-Suspended on unregistered processes: 1-7,24-26,28-31,48-55

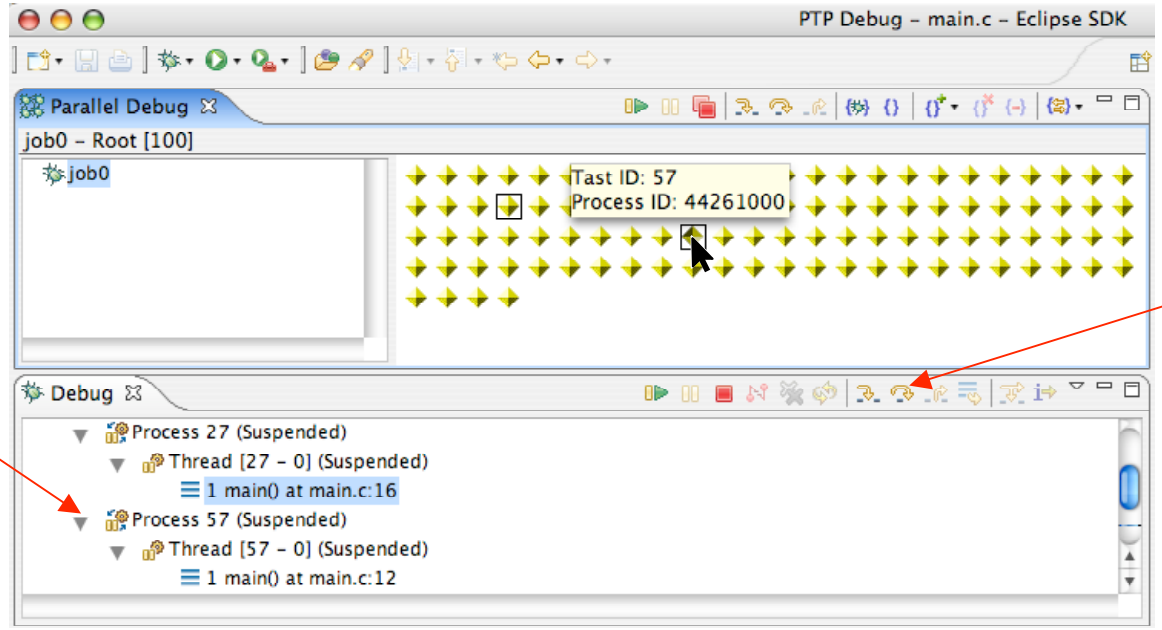
Parallel Debugging

- Process Registration
  - Process set commands apply to groups of processes
  - For finer control and more detailed information a process can be *registered*
  - Registered processes appear in the Debug view
  - Any number of processes can be registered
  - Processes can be registered or un-registered at any time

Parallel Debugging

- Register a process by double clicking on its process icon

Registered processes appear in Debug view



Debug commands apply to the currently selected process

- Un-register by double clicking on same icon

Parallel Debugging

- Putting it all together
  - Set a global breakpoint
  - Start 100 process job
  - When the breakpoint is reached
    - Create a process set containing processes 0-10 (set_1)
    - Create a process set containing processes 20-30 (set_2)
    - Single step all processes
    - Single step processes in set_1
    - Register a process
    - Single step the registered process
    - Select stack frame, view variables
  - Terminate execution

Parallel Tools Platform

- Future Plans (Parallel Debugging)
  - Scalability improvements (10-100K processes)
  - Additional architecture support (MPICH)
  - Program data visualization
    - Array viewer
    - Vector field viewer
    - Simple isosurface viewer
    - Distributed data viewer
  - Advanced debugging
    - Relative debugging
    - Replay/post-mortem debugging
    - MPI message debugging

Parallel Tools Platform

MPI DEVELOPMENT TOOLS

MPI Development Tools

- Tools to assist the development of MPI programs
- Initial contribution by IBM
  - Contact: Beth Tibbitts <tibbitts@us.ibm.com>
- Ongoing development effort
- Many of these techniques can be applied generally, not just MPI

MPI Development Tools

- Features currently include
  - Syntax highlighting of MPI constructs
  - Table display of all MPI constructs in code
  - Navigation to construct location
  - Content assist
  - API detail using mouse hover
  - Context sensitive help using F1/Help key

MPI Development Tools

- Main features of the MPI Development Tools

Syntax Highlighting

MPI Artifact Markers

Context Sensitive Help

Navigation Markers

MPI Artifact List

C/C++ - skosfile.c - Eclipse SDK

C/C++ Projects

test
Binaries
Includes
Debug
skosfile.c

```
    || (sendcnts == NULL)
    || (senddispls == NULL)
    || (recvbuf == NULL)
    || (recvcnts == NULL)
    || (recvdispls == NULL)) {
  return MPI_ERR_BUFFER;
}

if ((error_code = MPI_Comm_size(comm,&comm_size)) != MPI_SUCCESS) {
  return error_code;
}
MPI_Comm_rank(comm,&my_rank);
#define RR_ALLTOALL_TAG 11

MPI_Type_extent(sendtype,&send_type_size);
MPI_Type_extent(recvtype,&recv_type_size);

mpi_reqs = malloc(2 * comm_size * sizeof(MPI_Request));
mpi_stati = malloc(2 * comm_size * sizeof(MPI_Status));
NAssert( mpi_reqs != NULL && mpi_stati != NULL );

for (i = 0; i < comm_size; i++) {
  MPI_Isend(((char *)sendbuf) + (senddispls[i] * send_type_size),sendcnts[i],
        sendtype,i,RR_ALLTOALL_TAG,comm,mpi_reqs + i);
}
for (i = 0; i < comm_size; i++) {
  MPI_Irecv(((char *)recvbuf) + (recvdispls[i] * recv_type_size),recvcnts[i],
        recvtype,i,RR_ALLTOALL_TAG,comm,mpi_reqs + comm_size + i);
}
MPI_Waitall(2 * comm_size,mpi_reqs,mpi_stati);
```

Outline

DATA_LIST
list_start
list_end
list_last
number
(anon)
DL_NO_ERROR
DL_NO_ELEM
DL_NO_MEM
(anon)
END
START
LAST
append
init_list
add
read_ele
read_item_ele
item_addr
item_addr_at_item
number_of_elemen
is_end
is_start

Help

MPI_Comm_rank

int MPI_Comm_rank(MPI_Comm,

Returns the rank of the local task in the group associated with a communicator.

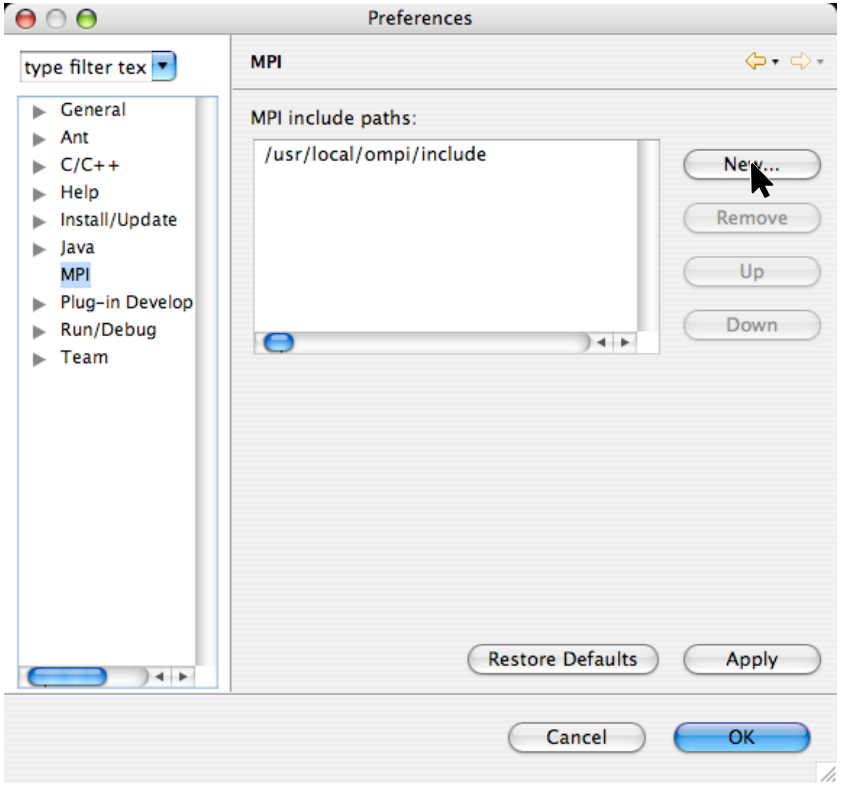
Go To:
All Topics Search
Related Topics Bookmarks

Problems | Console | Properties | MPI Artifact View

MPI Artifact	Filename	LineNo	Construct
MPI_MAX_PROCESSOR_NAME	skosfile.c	104	Constant
MPI_Irecv	skosfile.c	1833	Function Call
MPI_ANY_TAG	skosfile.c	1834	Constant
MPI_Wait	skosfile.c	1842	Function Call
MPI_Ssend	skosfile.c	1844	Function Call

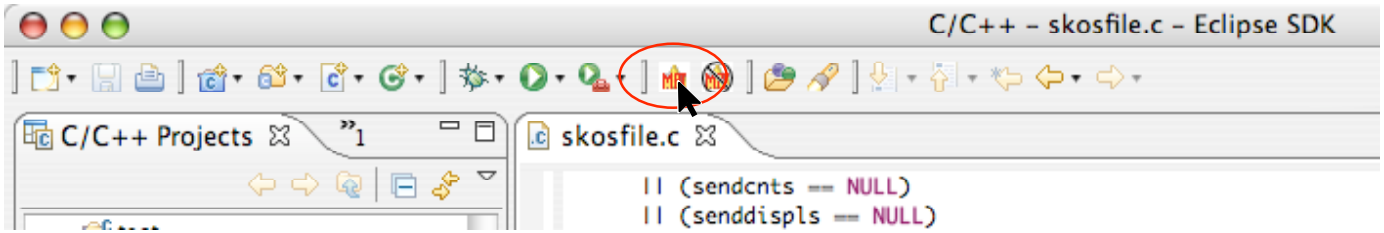
MPI Development Tools

- Configuring MPI Development Tools
  - Add include path(s) in MPI Preferences

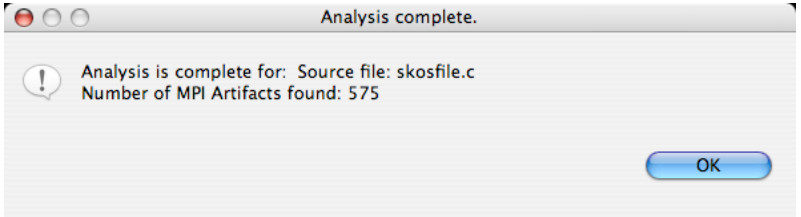


MPI Development Tools

- Analyzing source code
  - Select source file or folder and select analyze button



  - The number of artifacts found is reported when analysis is complete



MPI Development Tools

- Navigation
  - Double click on line in MPI Artifact View to navigate to the corresponding source line

Problems | Console | Properties | MPI Artifact View

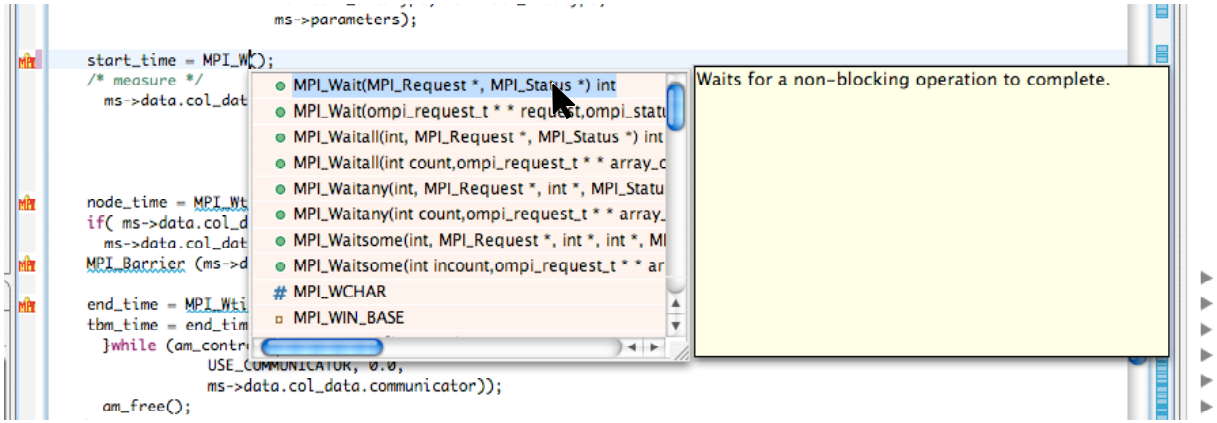
...	MPI Artifact	Filename	LineNo	Construct
	MPI_Wtime	skosfile.c	15577	Function Call
	MPI_Wtime	skosfile.c	15691	Function Call
	MPI_Wtime	skosfile.c	15698	Function Call

  - Artifacts can be sorted by MPI Artifact, Filename, Line No or Construct
  - Click on blue navigation marker to navigate to the corresponding region of code

```
                                ms->send_datatype,
                                ms->recv_datatype);
node_time = MPI_Wtime() - start_time;
if( ms->data.col_data.finalize_server_routine != NULL)
  ms->data.col_data.finalize_server_routine();
MPI_Barrier (ms->data.col_data.communicator);
```

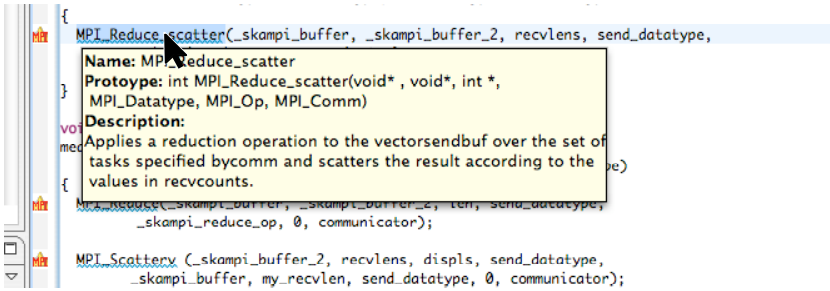
MPI Development Tools

- Content Assist
  - Control-space will suggest alternatives

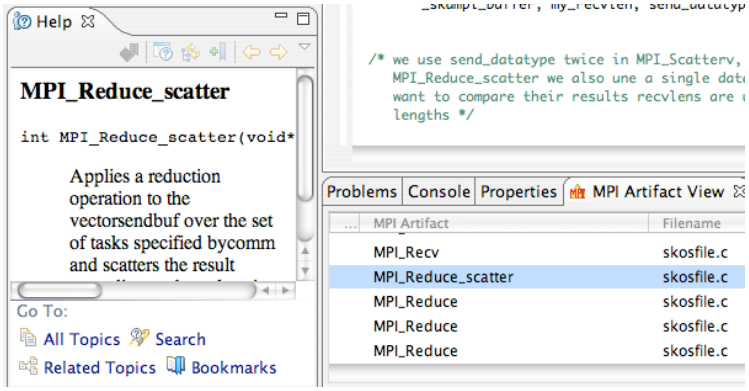


MPI Development Tools

- Context Sensitive Help
  - Hover over artifact will provide additional information

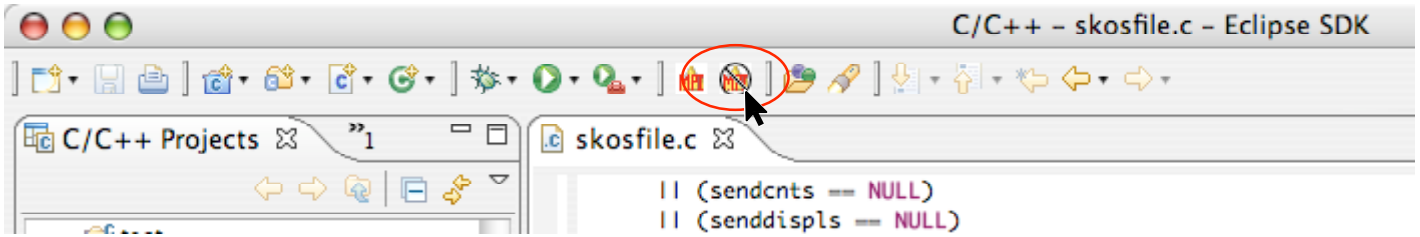


  - Pressing F1 (Windows) or Help (MacOSX) will provide more help



MPI Development Tools

- Remove MPI Annotation
  - Removes MPI artifact and navigation markers from editor window and MPI Artifact View



Parallel Tools Platform

- Future Plans (MPI Development Tools)
  - Static/dynamic analysis
    - Correctness
  - MPI code generation
    - From high level constructs (e.g. Co-array Fortran)
    - From TeX or other markup language
    - From data type definitions
  - Graphical representation
    - MPI communicators
    - MPI topology
  - Better integration with parallel debugger