

Module 5: Parallel Debugging

✦ Objective

- ✦ Learn the basics of debugging parallel programs

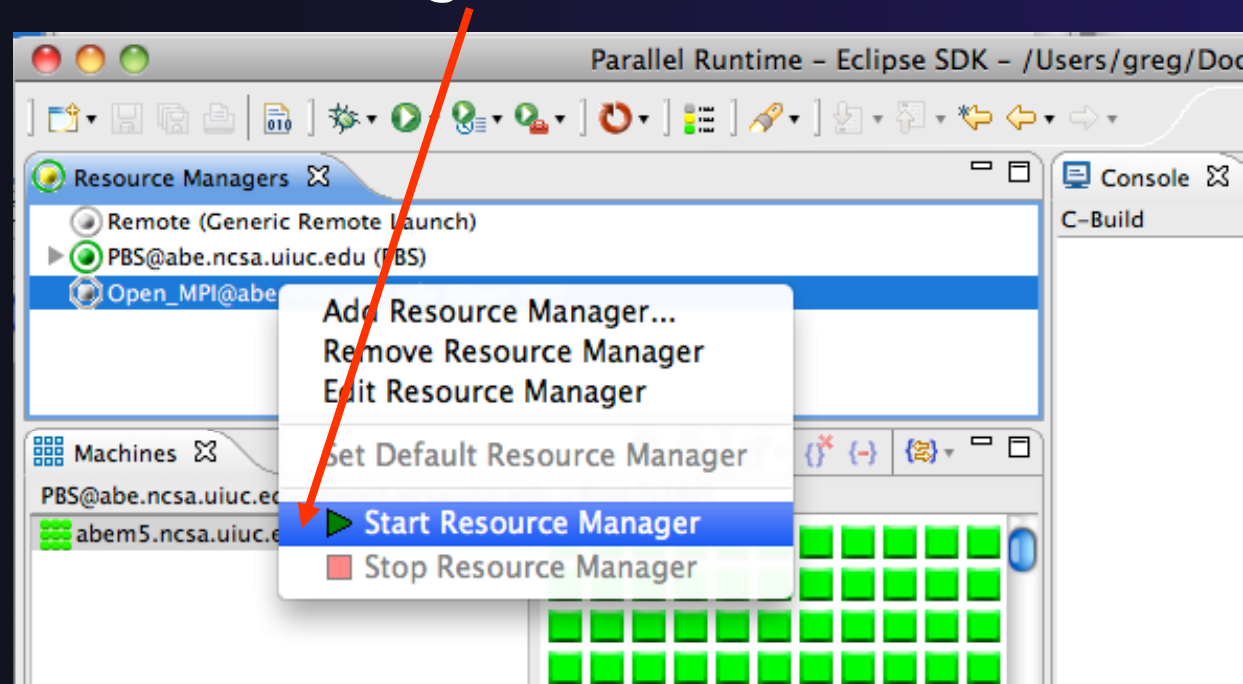
✦ Contents

- ✦ Launching a debug session
- ✦ The Parallel Debug Perspective
- ✦ Controlling sets of processes
- ✦ Controlling individual processes
- ✦ Parallel Breakpoints
- ✦ Terminating processes



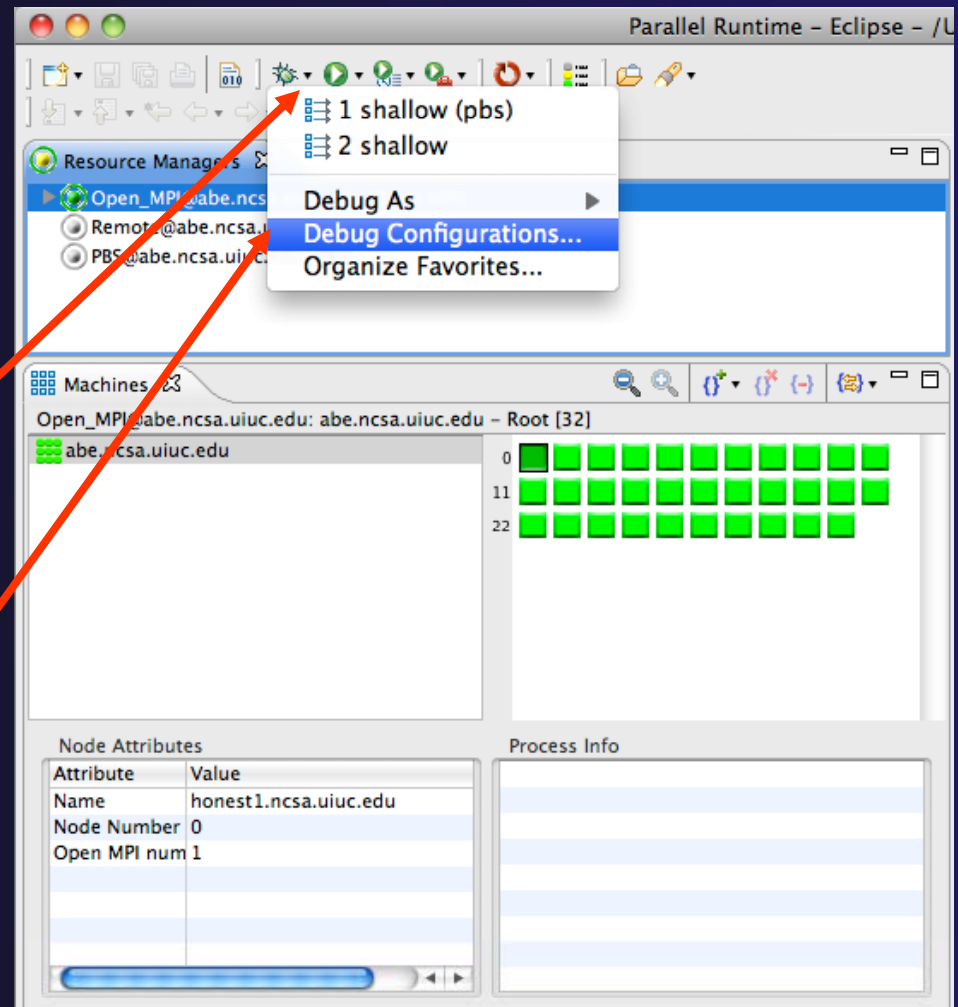
Start the Resource Manager

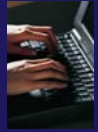
- ★ If the Open_MPI Resource manager is not already started (green icon), start it now:
- ★ Right-click on the resource manager and select **Start Resource Manager** from the menu



Create a Debug Configuration

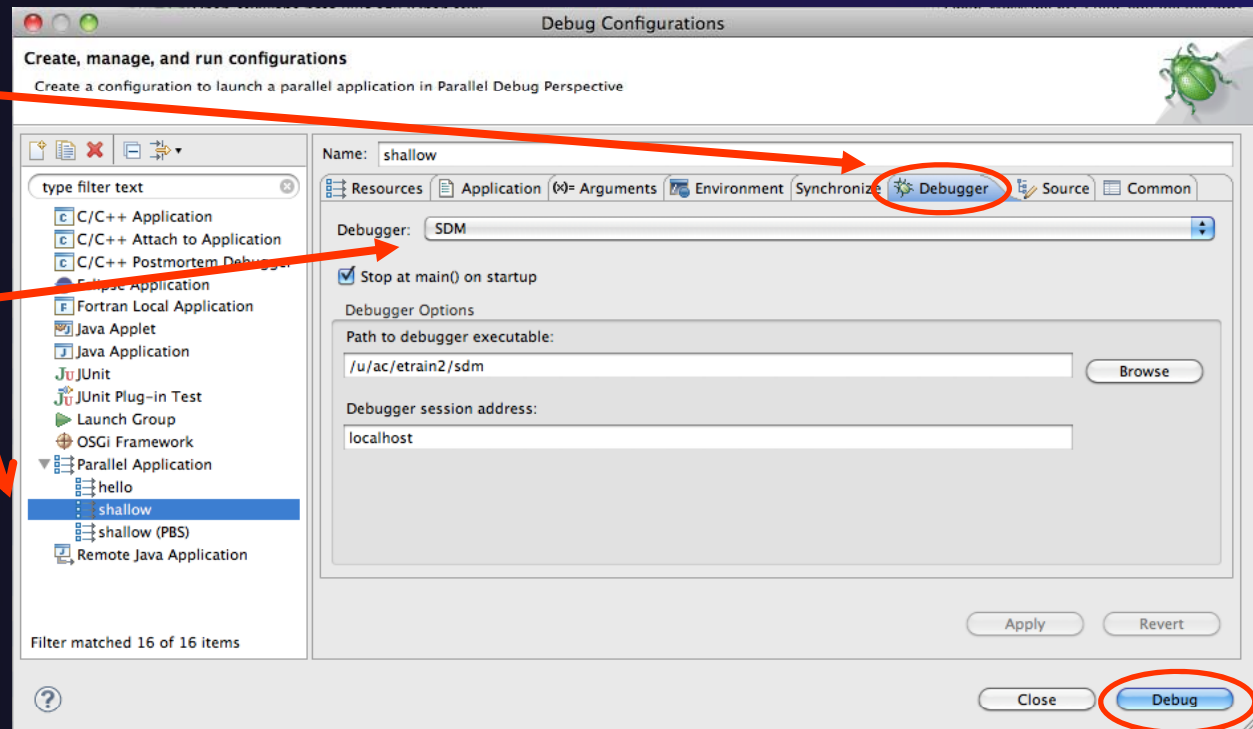
- ★ A debug configuration is essentially the same as a run configuration (like we used in modules 3 & 4)
- ★ We will re-use the existing configuration and add debug information
- ★ Use the drop-down next to the debug button (bug icon) instead of run button
- ★ Select **Debug Configurations...** to open the **Debug Configurations** dialog





Configure the Debugger Tab

- ★ Select **Debugger** tab
- ★ Select the **shallow** configuration
- ★ Make sure **SDM** is selected in the **Debugger** dropdown
- ★ Check the debugger path is correct
 - ★ Should be the path to the sdm executable on the remote system
- ★ Debugger session address should not need to be changed
- ★ Click on **Debug** to launch the program



The Parallel 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 screenshot displays the Eclipse IDE in the Parallel Debug perspective. The top toolbar includes icons for Parallel Debug, Parallel Runtime, and Remote C/C++. The main interface is divided into several panes:

- Parallel Debug:** Shows the job 'job0' and its processes.
- Debug:** Shows the thread 'Thread [1] (Suspended)' and the call stack for 'main0 main.c:38 4032ca'.
- Source:** Shows the source code for 'main.c' with a current line marker at line 38: `float pi=4.*(float)atan((double)1.);`.
- Variables:** Shows a list of variables and their values:

Name	Value
argc	1
argv	7fffffff658
pi	5.957805E-39
p	[0 - 18]
u	[0 - 18]
v	[0 - 18]
psi	[0 - 18]
- Outline:** Shows the project structure with files like 'math.h', 'mp.h', 'stdio.h', 'dec.h', and functions like 'worker(): void', 'setup_res(): MPI_Datatype*', 'main(int, char*())', 'update_global_ds(MPI_Datatype)', and 'setup_res(): MPI_Datatype*'.
- Console:** Shows the output of the application: 'Open_MPI@abe.ncsa.uiuc.edu:default:job0'.
- Memory/Problems:** Shows the memory usage and any problems encountered during debugging.

The Parallel 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

The screenshot displays the Eclipse IDE in the Parallel Debug perspective. The top toolbar includes icons for Breakpoint, Expressions, Variables, Signals, and Arrays. The Breakpoint view shows a single breakpoint set at line 38 of main.c. The Variables view displays the following data:

Name	Value
argc	1
argv	7fffffff658
pi	5.957805E-39
p	[0 - 18]
u	[0 - 18]
v	[0 - 18]
psi	[0 - 18]

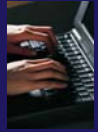
The Debug view shows the current process as '1 main() main.c:38 4032ca'. The source code editor displays the following code snippet:

```

32 MPI_Datatype * setup_res();
33
34 main (argc, argv)
35     int argc;
36     char * argv[];
37 {
38     float pi=4.*(float)atan((double)1.);
39     float p[n][n]; /* Pressure (or free surface height) */
40     float u[n][n]; /* Zonal wind */
41     float v[n][n]; /* Meridional wind */
42     float psi[n][n]; /* Velocity streamfunction */
43     float pold[n][n];
44     float uold[n][n];
45     float vold[n][n];
46     float h[n][n];
47     float dummy1[n];
48     float dummy2[n][n];
49     float tpi=pi+p;
50     float di=tpi/(float)m;
51     float dj=tpi/(float)n;
52

```

The Outline view on the right shows the project structure, including files like math.h, mp.h, stdio.h, decs.h, and functions like worker(), void, setup_res(), main(int, char**), and update_global_ds(MPI_Datatype).



Stepping All Processes

- ★ The buttons in the **Parallel Debug View** control groups of 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

Parallel Debug - shallow/main.c - Eclipse - /Users/greg/testing/workspa

Parallel Debug View: Open MPI@abe.ncsa.uiuc.edu: default: job0 Root [4] job0

Debug View: shallow [Parallel Application] Process 0 (Suspended) Thread [1] (Suspended) 1 main() main.c:50 4032f6

```

main.c
38 float pi=4.*(float)atan((double)1.);
39 float p[n][m]; /* Pressure (or free surface height) */
40 float u[n][m]; /* Zonal wind */
41 float v[n][m]; /* Meridional wind */
42 float psi[n][m]; /* Velocity streamfunction */
43 float polc[n][m];
44 float uolc[n][m];
45 float volc[n][m];
46 float h[n][m];
47 float z[n][m];
48 float dummy1[m];
49 float dummy2[n][m];
50 float tpi=pi+pi;
51 float di=tpi/(float)m;
52 float dj=tpi/(float)n;
53 int i, j, chunk_size, nxt, prv;
54
55 int master_packet[4];
56 float p_start[m];
57 float u_start[m];
58 float v_start[m];
  
```




Stepping An Individual Process

- ★ The buttons in the **Debug view** are used to control an individual process, in this case process 0
- ★ 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 screenshot displays the Eclipse IDE interface for debugging a parallel application. The top toolbar includes various debugging actions. The 'Parallel Debug' view shows a tree structure with 'Job0' containing 'Process 0 (Suspended)' and 'Thread [1] (Suspended)'. The 'Debug' view shows the 'Step Over' button (a right-pointing arrow with a red outline) highlighted. The 'main.c' source code is visible at the bottom, with line 51 highlighted in blue, indicating the current execution point for process 0. The code includes variable declarations for pressure, wind, and streamfunction, and a calculation for d_i .

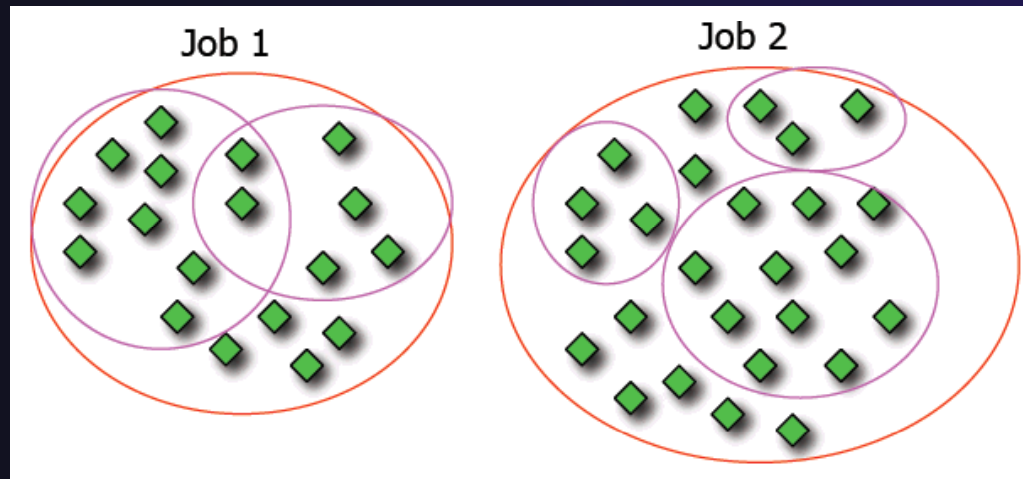
```

38 float pi=4.*(float)atan((double)1.);
39 float p[n][m]; /* Pressure (or free surface height) */
40 float u[n][m]; /* Zonal wind */
41 float v[n][m]; /* Meridional wind */
42 float psi[n][m]; /* Velocity streamfunction */
43 float pold[n][m];
44 float uold[n][m];
45 float vold[n][m];
46 float h[n][m];
47 float z[n][m];
48 float dummy1[m];
49 float dummy2[n][m];
50 float tpi=pi+pi;
51 float di=tpi/(float)m;
52 float dj=tpi/(float)n;
53 int i, j, chunk_size, nxt, prv;
54
55 int master_packet[4];
56 float p_start[m];
57 float u_start[m];

```

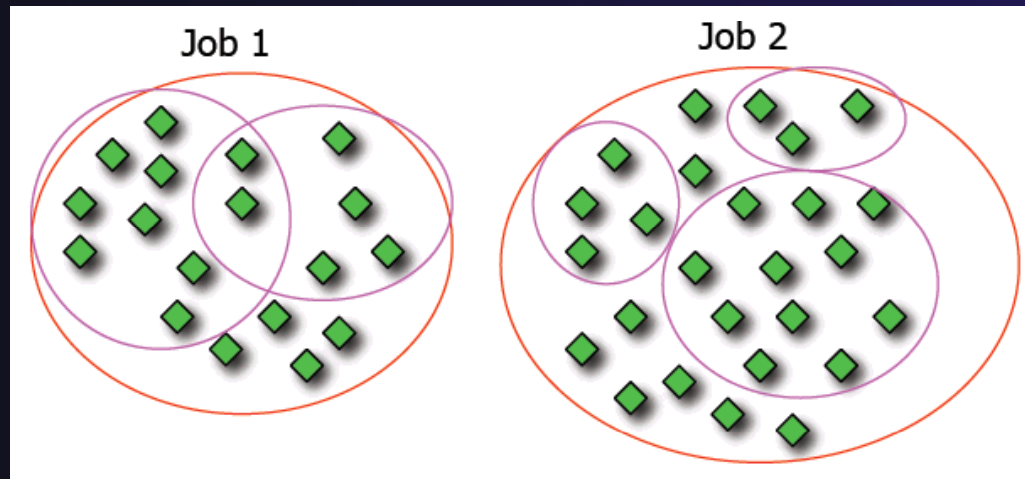
Process Sets (1)

- ★ 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



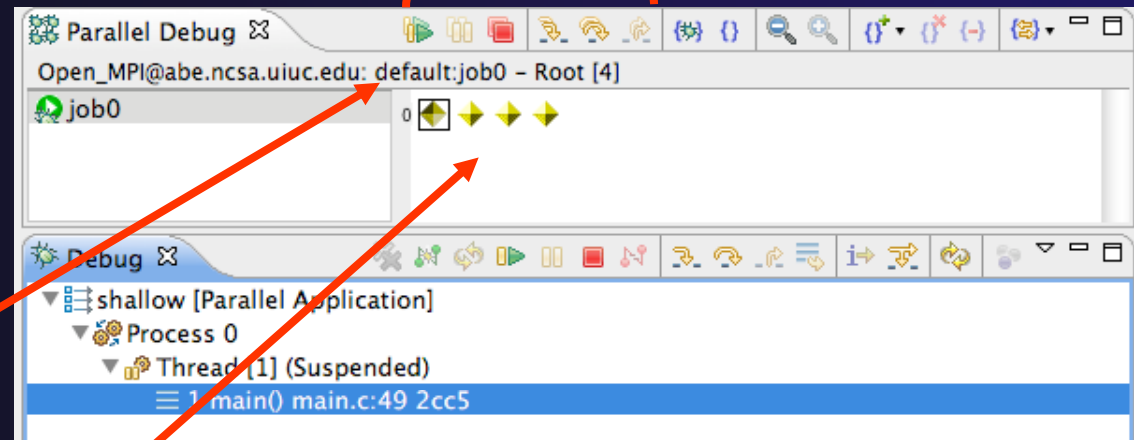
Process Sets (2)

- ★ 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

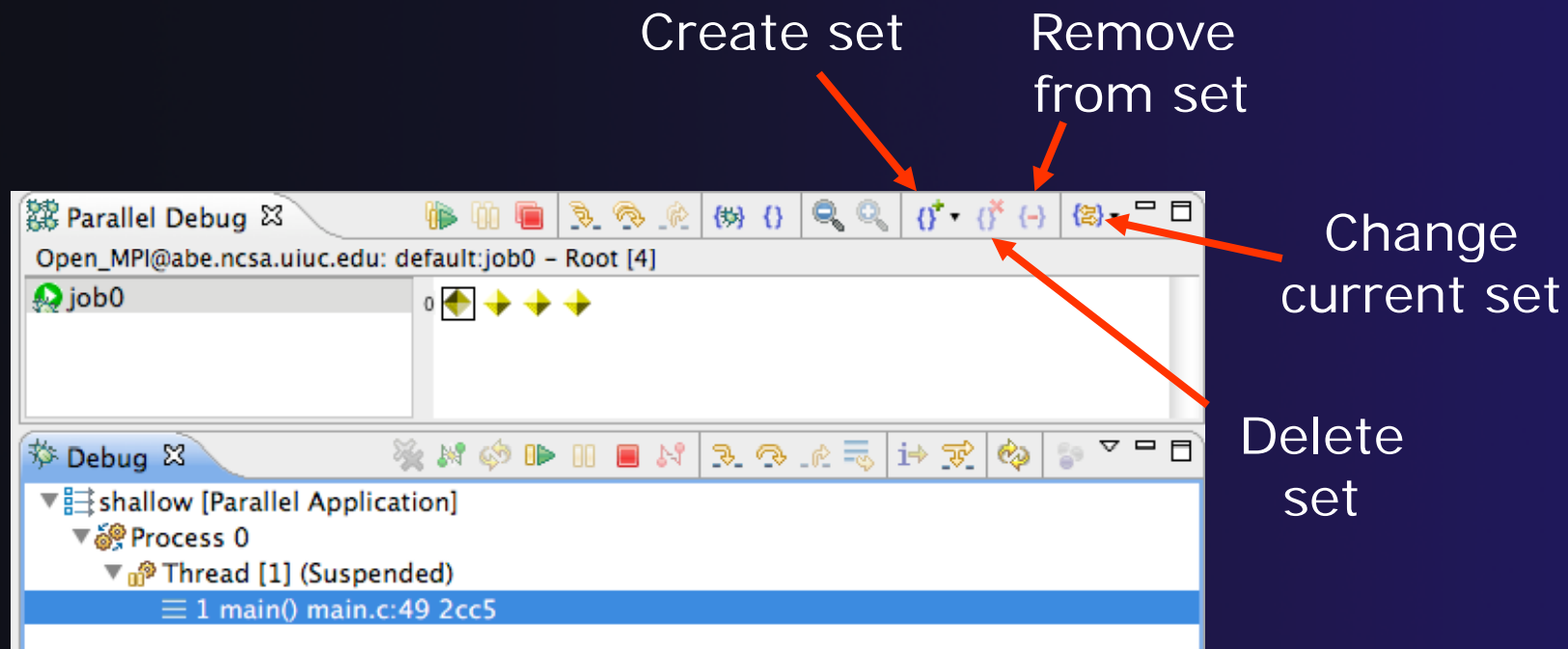
- ★ Debug operations on the **Parallel Debug view** toolbar always apply to the current set:
 - ★ Resume, suspend, stop, step into, step over, step return
- ★ The current process set is listed next to job name along with number of processes in the set
- ★ The processes in process set are visible in right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



Root set = all processes

Managing Process Sets

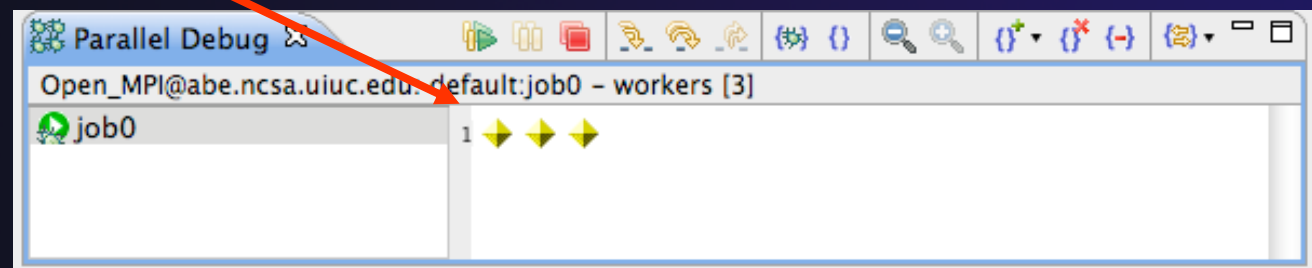
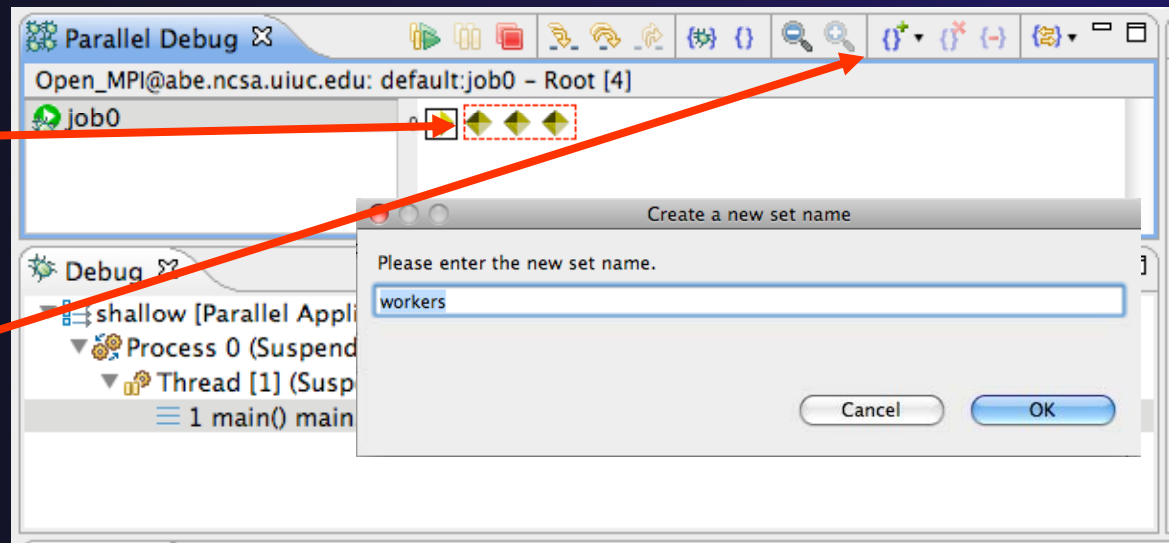
- ★ The remaining icons in the toolbar of the **Parallel Debug view** allow you to create, modify, and delete process sets, and to change the current process set



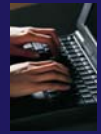


Creating A New Process Set

- ★ 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 see only the first current line marker move
- ★ Step a couple more times
- ★ You should see two line markers, one for the single master process, and one for the 3 worker processes

```

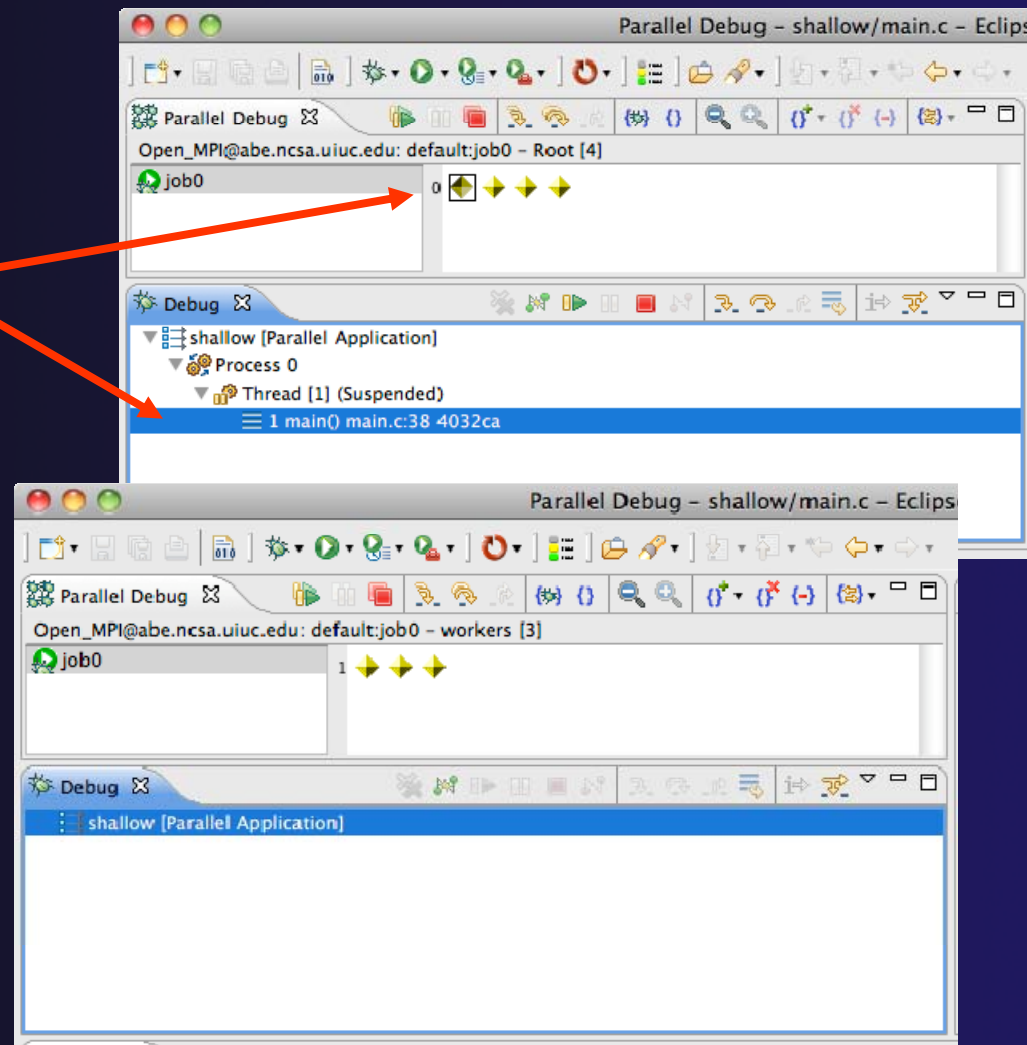
main.c
49 float dummyz[n][m];
50 float tpi=pi+pi;
51 float di=tpi/(float)m;
52 float dj=tpi/(float)n;
53 int i, j, chunk_size, nxt, prv;
54
55 int master_packet[4];
56 float p_start[n];
57 float u_start[n];
58 float v_start[n];
59 float psi_start[m];
60 float pold_start[m];
61 float uold_start[m];
62 float vold_start[m];
63 int proc_cnt;
64 int tid;
65 MPI_Datatype * res_type;
66
67 MPI_Init(&argc, &argv);
68 MPI_Comm_size(MPI_COMM_WORLD, &proc_cnt);//hello
69 MPI_Comm_rank(MPI_COMM_WORLD, &tid);
  
```

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

Process Registration (2)

- ✦ By default, process 0 was registered when the debug session was launched
- ✦ Registered processes are surrounded by a box and shown in the Debug view
- ✦ The Debug view only shows registered processes in the current set
- ✦ Since the "workers" set doesn't include process 0, it is no longer displayed in the Debug view



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

The screenshot displays the Parallel Debug interface for a C application named 'shallow/main.c'. The interface is divided into several panes:

- Parallel Debug View:** Shows a group of processes labeled 'job0' with a count of 3. A red arrow points to the 'register' button (a green square with a white plus sign) next to the process icon.
- Debug View:** Shows the execution state of the application. It is currently at 'Process 3 (Suspended)' with 'Thread [1] (Suspended)'. The selected thread is '1 main() main.c:67 403335'. A red arrow points to this thread, with the text 'Individual (registered) processes' next to it.
- Code Editor:** Shows the source code for 'main.c'. The line 'MPI_Init(&argc, &argv);' is highlighted in blue, indicating the current execution point.

Additional text annotations in red:

- 'Groups (sets) of processes' points to the 'job0' group in the Parallel Debug view.
- 'Individual (registered) processes' points to the selected thread in the Debug view.

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

```

main.c
int proc_cnt;
int tid;
MPI_Datatype * res_type;

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &proc_cnt);
MPI_Comm_rank(MPI_COMM_WORLD, &tid);

if ( proc_cnt < 2 )
{
    fprintf(stderr, "must have at least 2 processes, not %d\n", proc_cnt);
    MPI_Finalize();
    return 1;
}

```



Multiple processes marker



Registered process marker



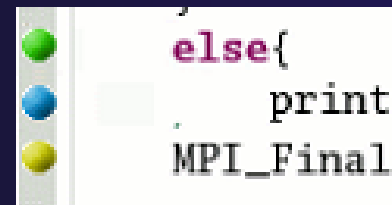
Un-registered process marker



Multiple markers at this line
 -Suspended on unregistered process: 2
 -Suspended on registered process: 1

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





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
- ★ The breakpoint is displayed on the marker bar

```

69 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
70
71 fprintf(stdout, "my rank is %d\n", tid);
72
73 if ( proc_cnt < 2 )
74 {
75     fprintf(stderr, "must have at least 2 processes, not %d\n", proc_cnt);
76     MPI_Finalize();
77     return 1;
78 }
79
80 if ( (n % (proc_cnt - 1)) != 0 )
81 {
82     if ( tid == 0 )
83         fprintf(stderr, "(number of processes - 1) must be a multiple of %d\n", n);
84
85     MPI_Finalize();
86     return 1;
87 }
88

```

Hitting the Breakpoint



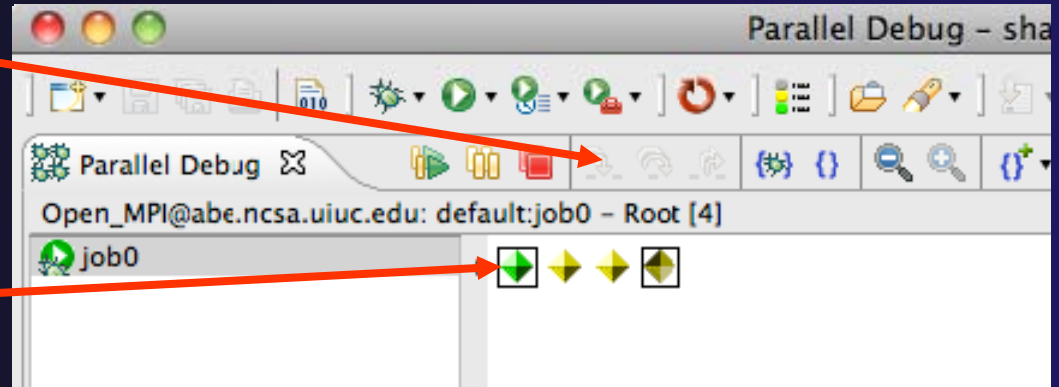
- ✦ Switch back to the **Root** set by clicking on the **Change Set** button
- ✦ Click on the **Resume** button in the **Parallel Debug** view
- ✦ In this example, the three worker processes have hit the breakpoint, as indicated by the yellow process icons and the current line marker
- ✦ Process 0 is still running as its icon is green
- ✦ Processes 1-3 are suspended on the breakpoint

```
Parallel Debug - shallow/main.c - Eclipse - /Users/greg/testing/v
Open_MPI@abe.ncsa.nc.edu: default:job0 - Root [4]
Job0
0
Debug
shallow [Parallel Application]
  Process 3 (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() main.c:80 4033c4
    Thread [3] (Suspended)
    Thread [2] (Suspended)
  Process 0
    Thread [1] (Running)
main.c
74 {
75     fprintf(stderr, "must have at least 2 processes, not %d\n", proc_cnt);
76     MPI_Finalize();
77     return 1;
78 }
79
80 if ( ( n % ( proc_cnt - 1) ) != 0 )
81 {
82     if ( tid == 0 )
83         fprintf(stderr, "(number of processes - 1) must be a multiple of %d\n", n);
84
85     MPI_Finalize();
86     return 1;
87 }
88
89 if (tid != 0) {
90     worker();
91     MPI_Barrier(MPI_COMM_WORLD);
92     MPI_Finalize();
93 } else {
```

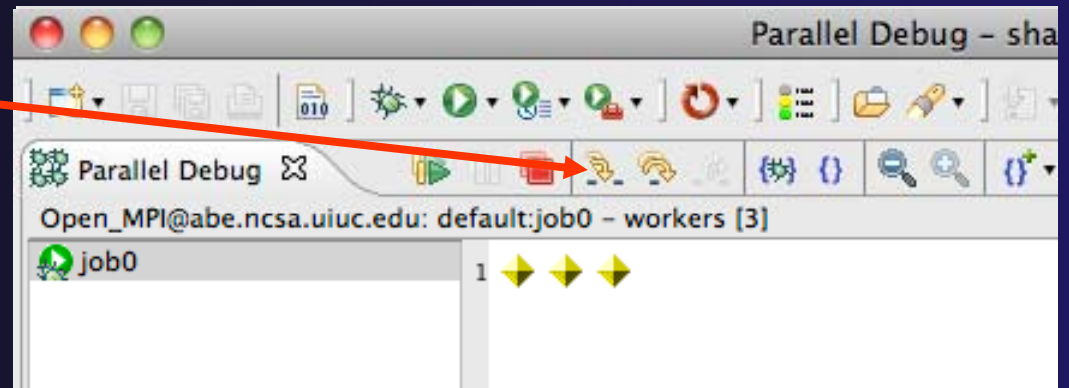



More On Stepping

- ★ 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



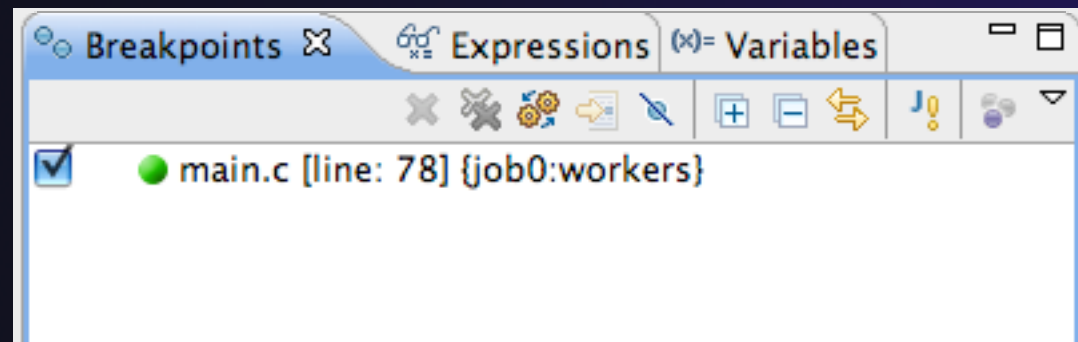
- ★ Switch to the set of suspended processes (the **workers** set)
- ★ You will now see the **Step** buttons become enabled

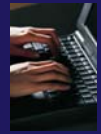




Breakpoint Information

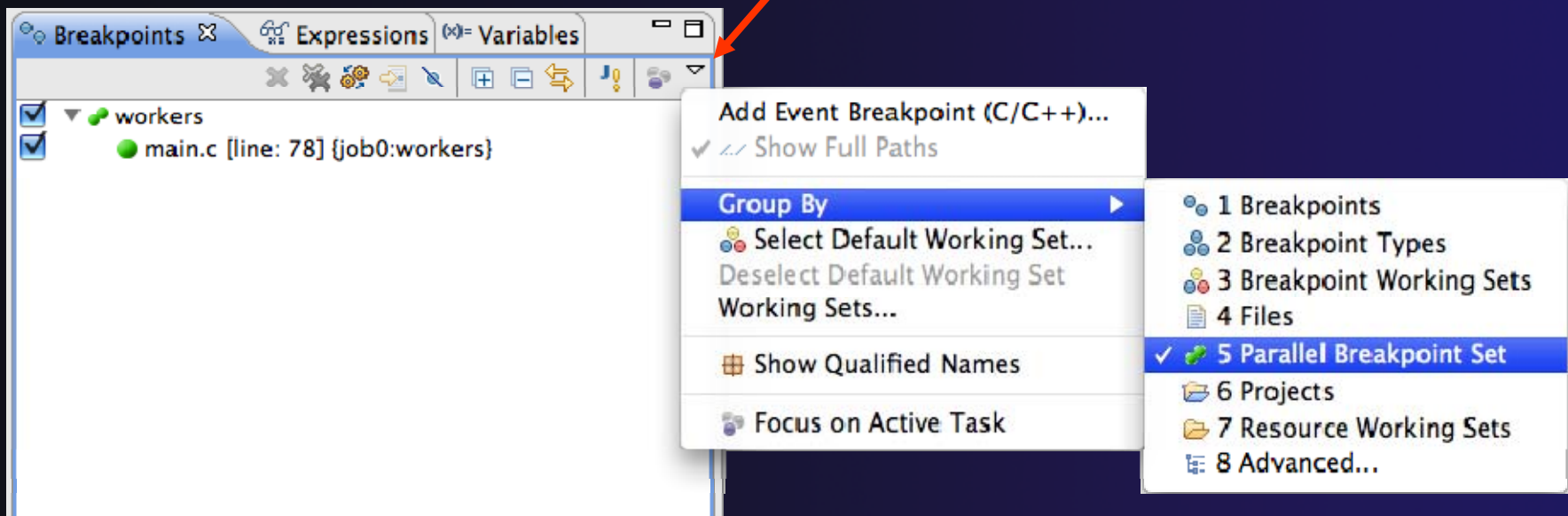
- ✦ Hover over breakpoint icon
 - ✦ Will show the sets this breakpoint applies to
- ✦ Select **Breakpoints** view
 - ✦ Will show all breakpoints in all projects





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

```
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

