Extending CDT To Debug Parallel Programs

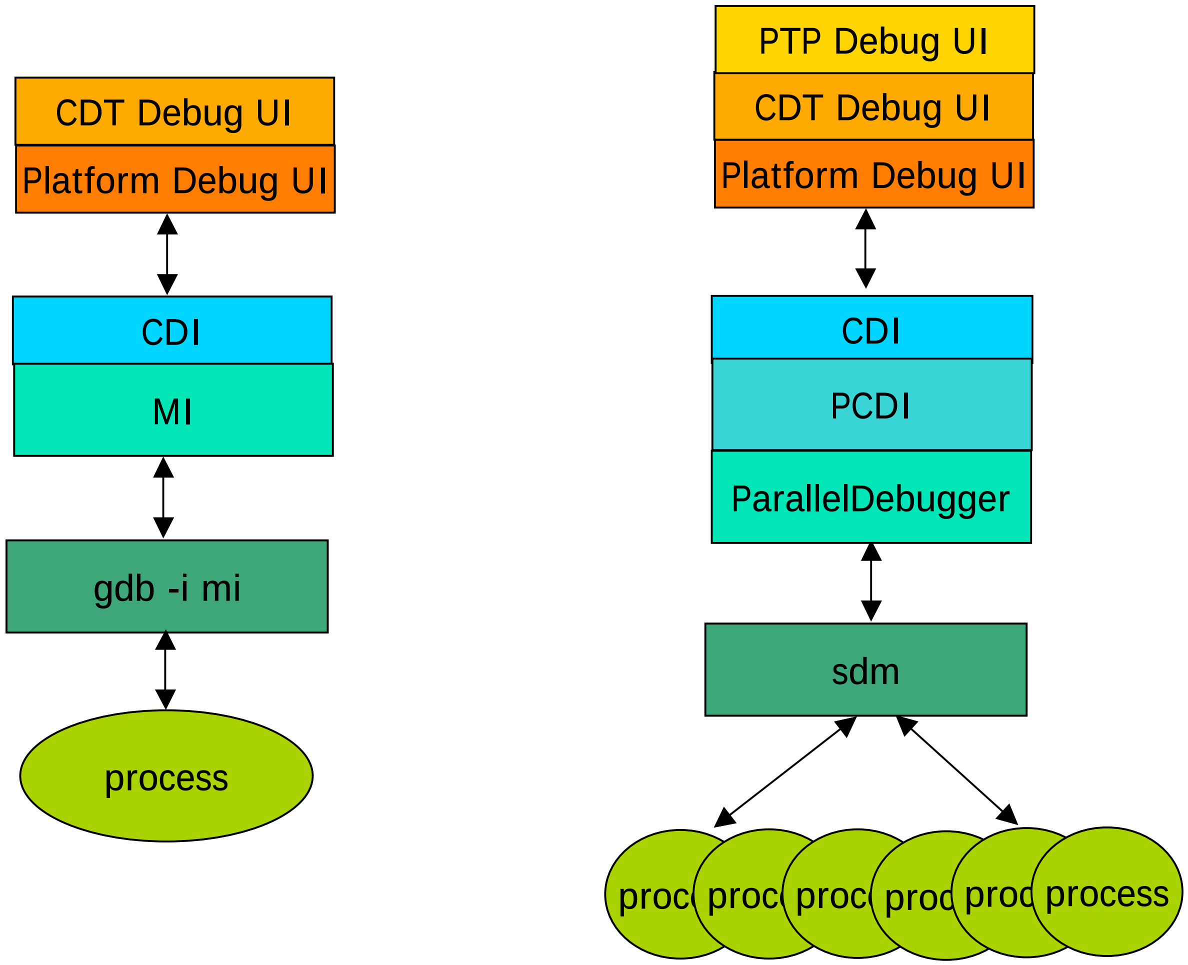
Greg Watson
Los Alamos National Laboratory

Clement Chu & Donny Kurniawan
Monash University

The Problem

- Two commercial parallel debuggers (TotalView & DDT)
  - Cost
  - Innovation
- No open-source parallel debuggers
  - mpigdb (gdb multiplexer)
  - gdb
  - printf
- No parallel debugger integrated with Eclipse
- Debuggers are *hard*, parallel debuggers *harder*

PTP Debug Architecture



Extensions to CDT Model

- CDT MI not used
  - No parallel support
- Many CDI interfaces extended
  - Anything that deals with a process
  - Added process set to minimize iteration over processes
- Added new level to model
  - Processes
  - Target -> Process
  - Thread remains (only single threaded currently supported)

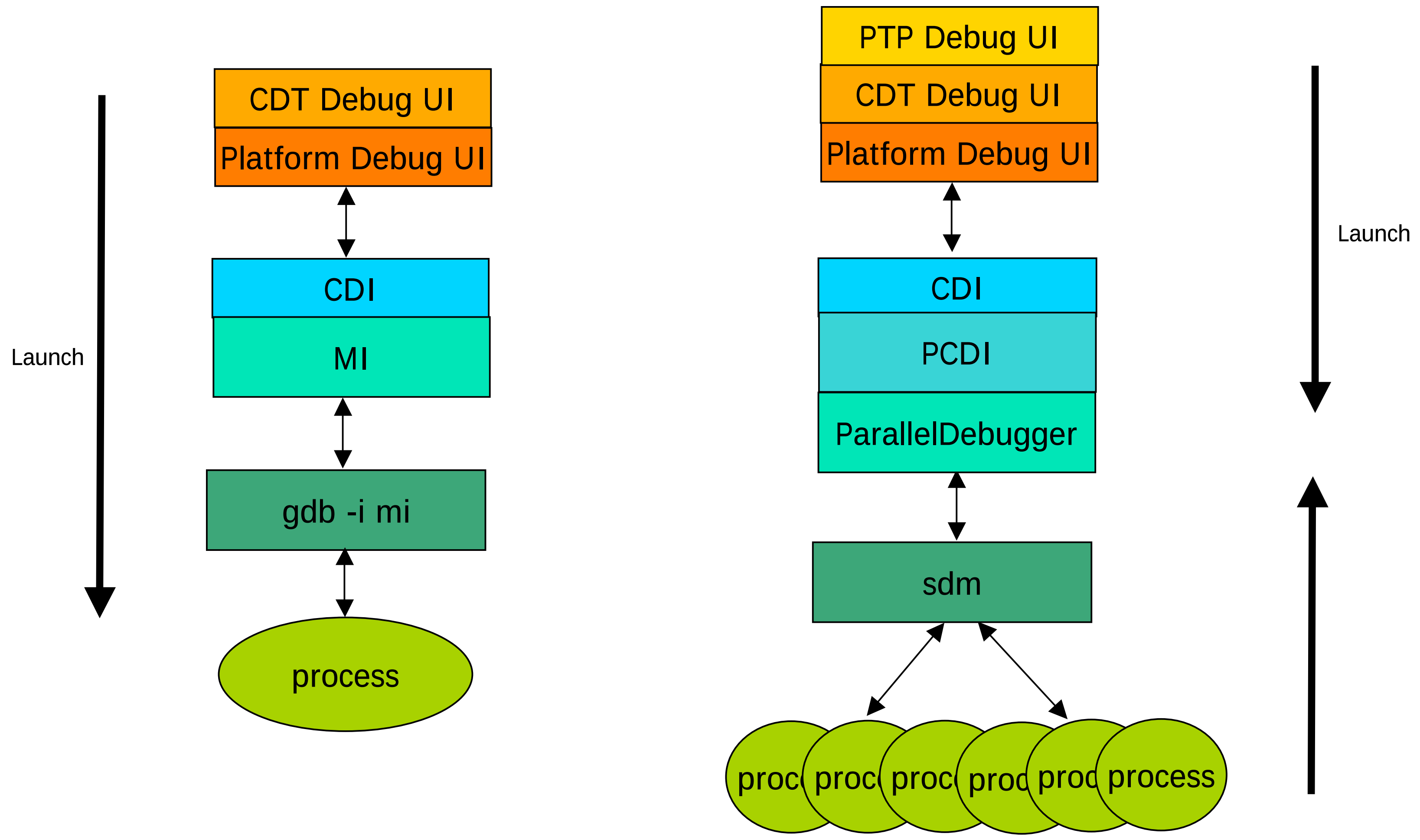
Extensions to CDT UI

- Tried to preserve as much as possible
- Need to deal with many processes
  - Performance
  - Complexity
- "Drill down" architecture
  - High level deals with groups of processes
  - User must explicitly request more information
- Breakpoints
- Current instruction pointer

Launching

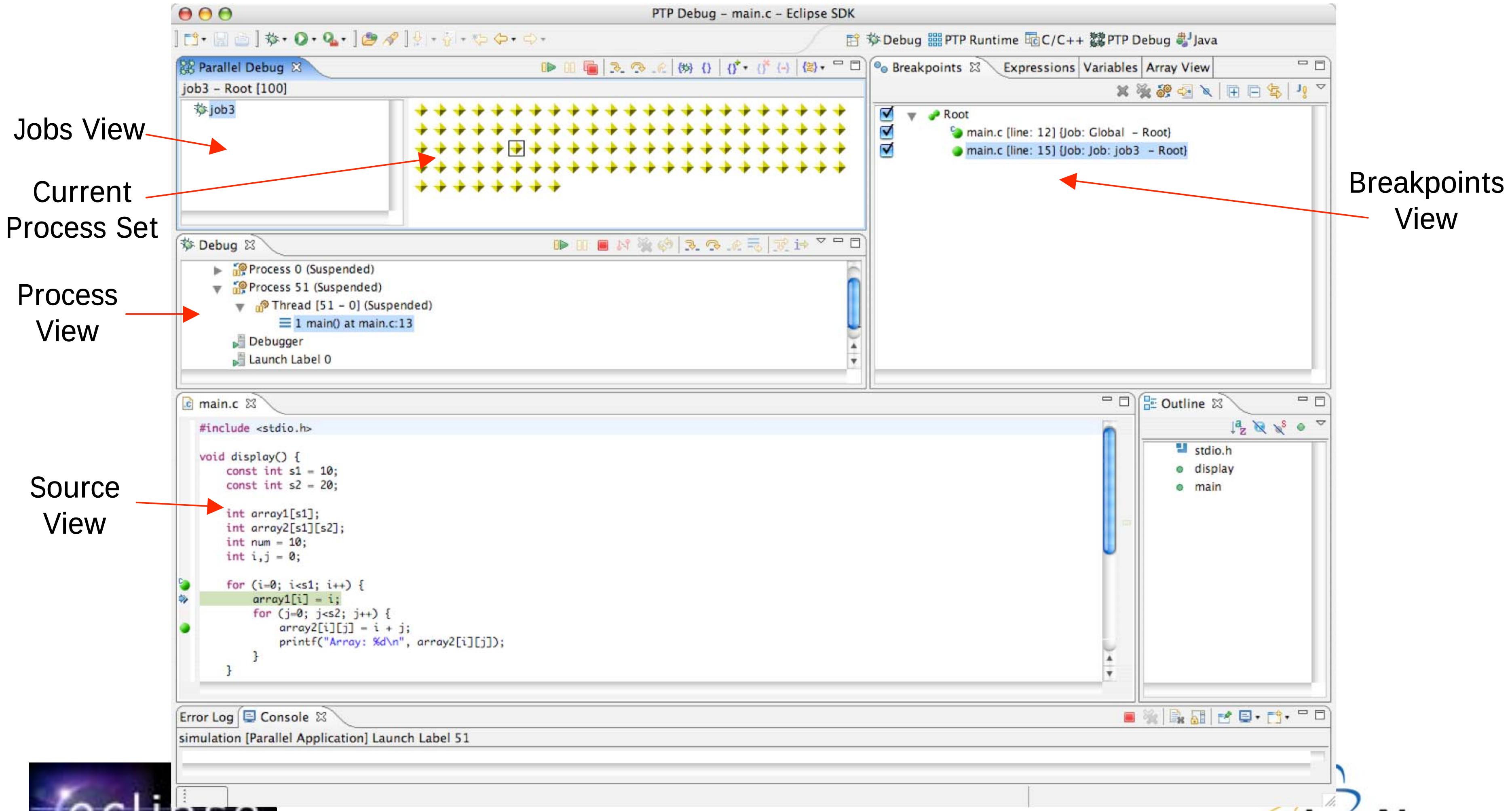
- Hard
- Does not fit CDI model

PTP Debug Architecture



UI Features

- Main features of the PTP Debug perspective



job3 – Root [100]

job3

```
Process 0 (Suspended)
Process 51 (Suspended)
  Thread [51 – 0] (Suspended)
    1 main() at main.c:13
Debugger
Launch Label 0
```

```
Root
  main.c [line: 12] {Job: Global – Root}
  main.c [line: 15] {Job: Job: job3 – Root}
```

```
#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]);
        }
    }
```

```
stdio.h
display
main
```

simulation [Parallel Application] Launch Label 51

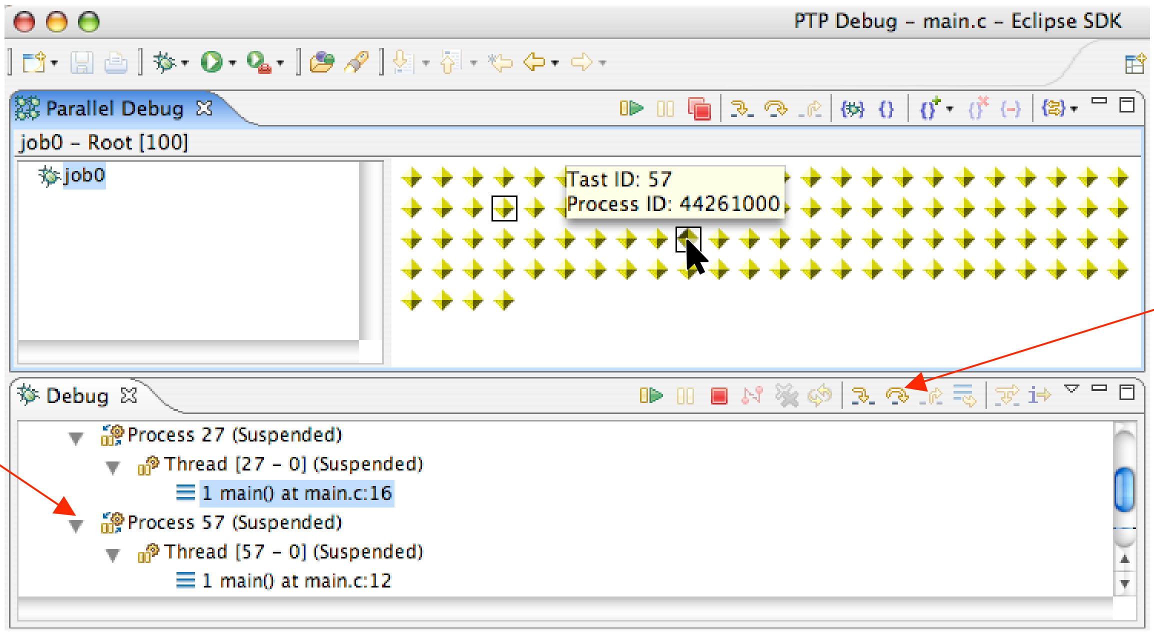
UI Features

- 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

UI Features

- Register a process by double clicking on its process icon

Registered processes appear in Debug view



Process 27 (Suspended)
Thread [27 – 0] (Suspended)
1 main() at main.c:16
Process 57 (Suspended)
Thread [57 – 0] (Suspended)
1 main() at main.c:12

Debug commands apply to the currently selected process

- Un-register by double clicking on same icon

UI Features

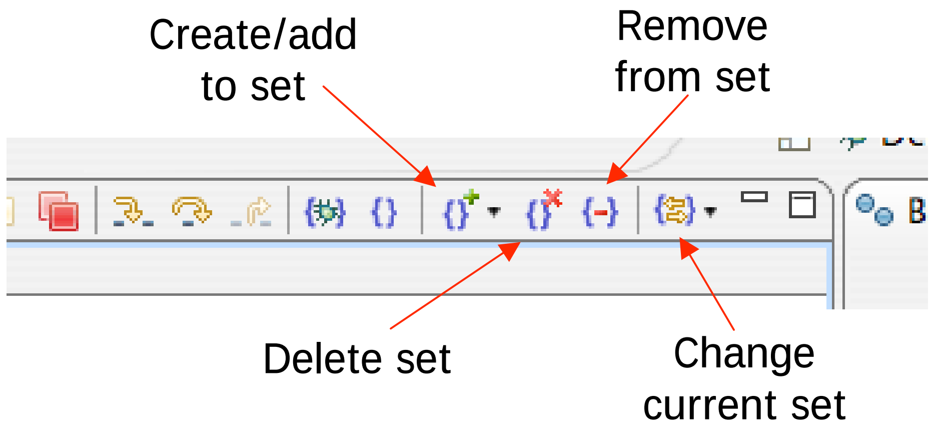
- Process sets
  - Traditional debuggers apply operations to a single processes
  - Parallel debugging operations apply to single process *or to arbitrary collections of processes*

Definition:

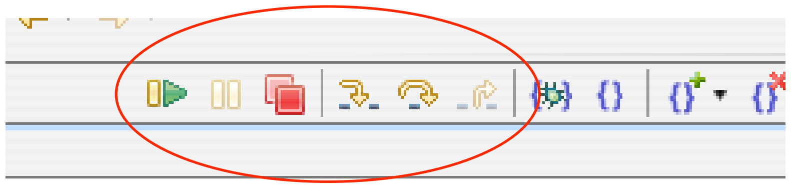
- A *process set* is a means of simultaneously referring to one or more processes

UI Features

- Process set operations

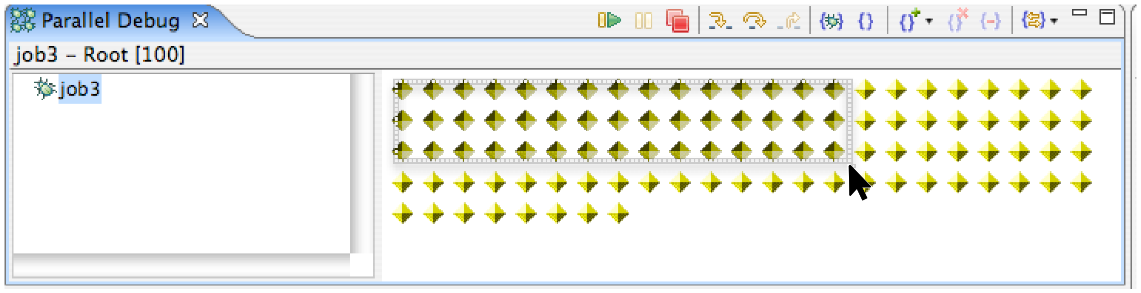


- Debug operations always apply to the current set

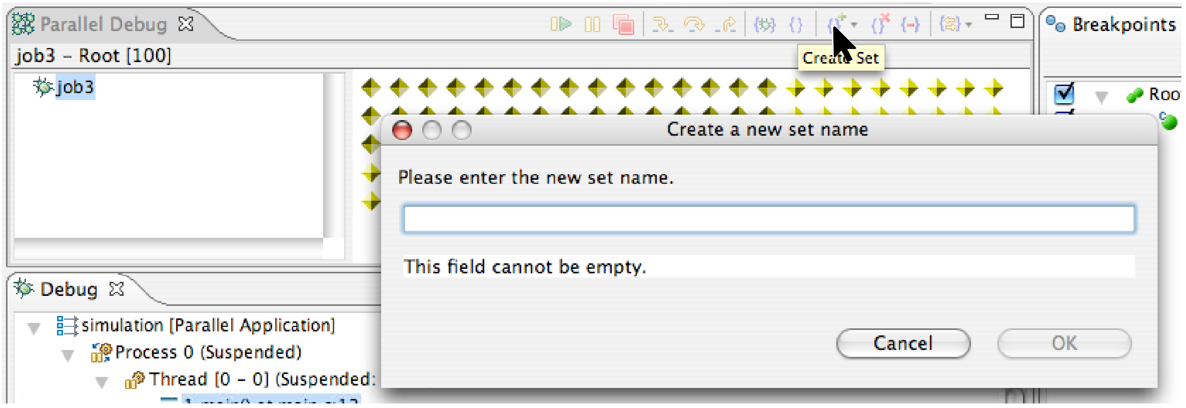


UI Features

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



  - Select create process set button and enter a name

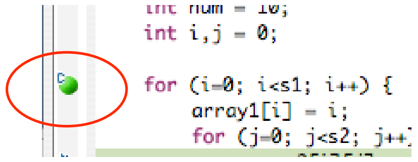


Please enter the new set name.

This field cannot be empty.

UI Features

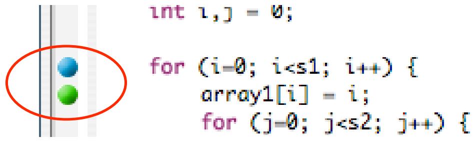
- Breakpoints
  - There are two main types of breakpoints
  - *Global breakpoints*
    - Apply to *all* processes in *any* job



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

for (i=0; i<s1; i++) {
    array1[i] = i;
    for (j=0; j<s2; j++
```

  - *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

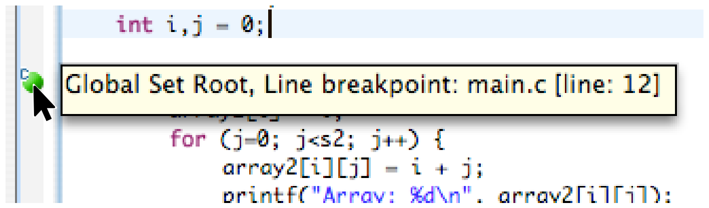


```
int i,j = 0;

for (i=0; i<s1; i++) {
    array1[i] = i;
    for (j=0; j<s2; j++) {
```

UI Features

- Breakpoint information
  - Hover over breakpoint to see more information

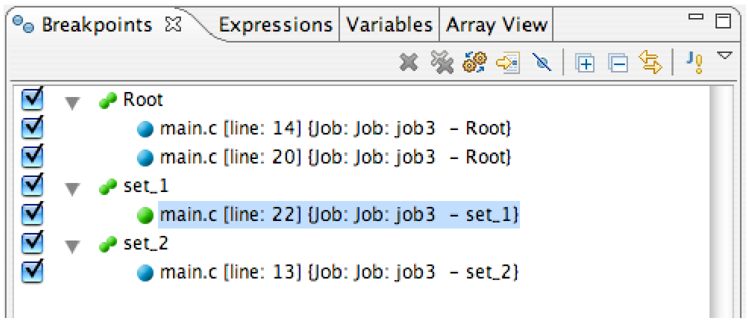


```
int i,j = 0;

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

Global Set Root, Line breakpoint: main.c [line: 12]

  - Use Breakpoints tab to see all breakpoints



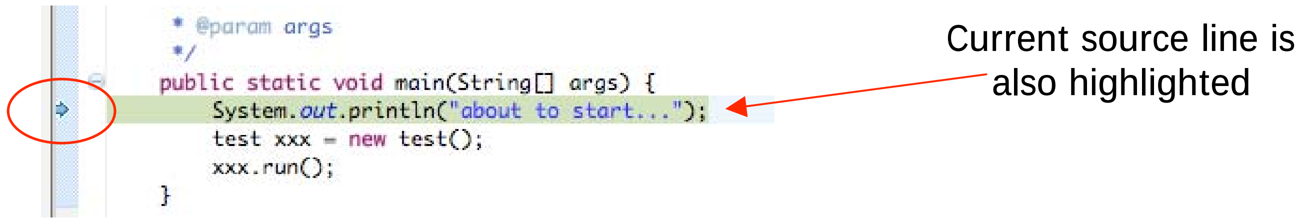
```
☑ ▼ Root
☑       main.c [line: 14] {Job: Job: job3 - Root}
☑       main.c [line: 20] {Job: Job: job3 - Root}
☑ ▼ set_1
☑       main.c [line: 22] {Job: Job: job3 - set_1}
☑ ▼ set_2
☑       main.c [line: 13] {Job: Job: job3 - set_2}
```

UI Features

- 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*

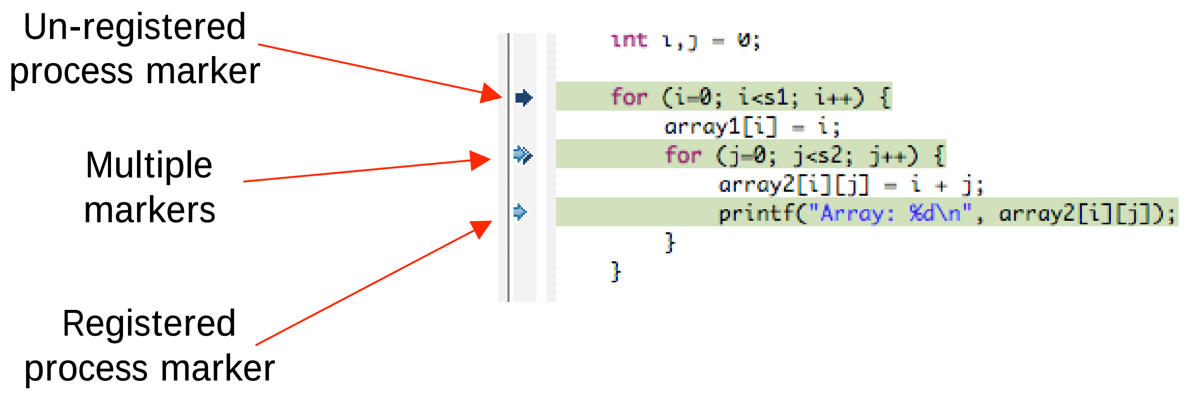
UI Features

- 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();
    }
```

- Multiple instruction pointers in PTP debugger

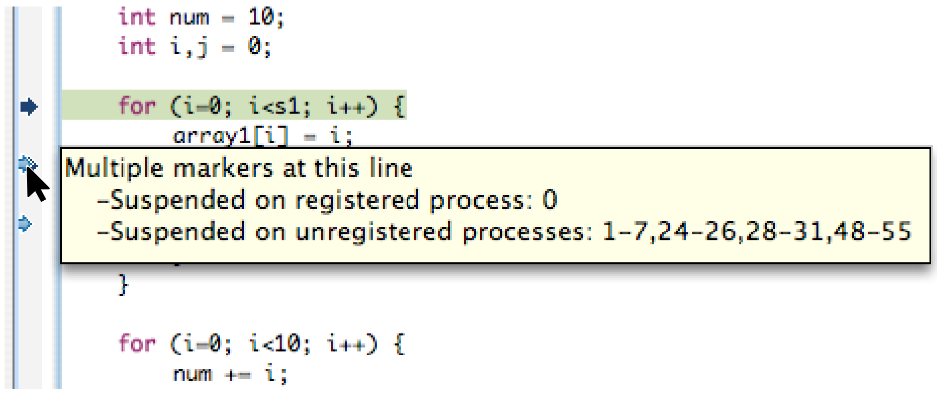


```
    int i,j = 0;

    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]);
        }
    }
```

UI Features

- Hovering over instruction pointer provides additional information



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

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

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

```
}

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

Demo

Parallel Debugging

- Future Plans
  - Scalability improvements (10-100K processes)
  - Additional architecture support (e.g. 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